Response Under 37 CFR § 1.116
Expedited Procedure - Examining Group 2157
PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appl. No.	:	09/489,759	Confirmation No. 6224
Applicant	:	Brian STYLES	
Filed	:	January 21, 2000	
TC/A.U.	:	2157	
Examiner	:	Saleh NAJJAR	
Docket No.	:	570-A00-001	
Customer No.	:	23334	

37 C.F.R. 1.131 AFFIDAVIT

I, the undersigned, the inventor of the above-referenced patent application, hereby declare the following:

1) The pending claims of our above identified patent invention were rejected under 35 U.S.C. 102(e) and 35 U.S.C. §103(a) based on the prior art reference of Bourke-Dunphy et al. (U.S. 6,449,642) with a effective filing date of September 15, 1998 based (hereinafter referred to as "Bourke").

2) The invention described in the above-referenced patent application was reduced to a writing by a software contractor Charles B. Bucklew that I hired prior to the September 15, 1998 date of Bourke. In particular, a partial listing of software modules and the contents of three source code modules (splashscreen.h; scriptm.cpp; and scriptmdlg.cpp) each with dates of original creation prior to September 15, 1998 are attached hereto. This code has been diligently worked on to carry out this invention from prior to September 15, 1998 through the January 21, 1999, after which date, the first public release of the code was made. This product has enjoyed great commercial success and has led to the formation of the company ScriptLogic Corporation in Boca Raton, Florida with sales of over 10 million dollars in 2003.

I, the undersigned, declare all of the above statements are made on our own knowledge, the above statements are true and correct, and the above statements are made on information that we believe to be true. I understand that false statements or concealment in obtaining a patent will subject us to fine and/or imprisonment or both (18 U.S.C. §1001) and may jeopardize the validity of the above identified patent application or any application issuing therefrom.

Brian Styles

June 30, 2004

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appl. No.	:	09/489,759	Confirmation No. 6224
Applicant	:	Brian STYLES	
Filed	:	January 21, 2000	
TC/A.U.	:	2157	
Examiner	:	Saleh NAJJAR	
Docket No.	:	570-A00-001	
Customer No.	:	23334	

37 C.F.R. 1.131 AFFIDAVIT

I, the undersigned, hereby declare the following:

1) My name is Charles B. Bucklew.

2) I am 24 years of age.

3) I reside at 11096 N.W. 9th Court, Plantation, Florida 33324

4) I am a citizen of the United States of America.

5) I currently work as a programmer for ScriptLogic Corporation.

6) I am the co-inventor on four pending patents applications.

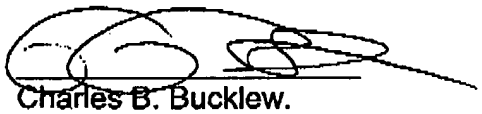
7) I have 10 years of experience working as a programmer.

8) I have reviewed the above-identified application and Bourke-Dunphy et al. (U.S. 6,449,642) with a effective filing date of September 15, 1998 based (hereinafter referred to as "Bourke").

8) I have reviewed the above-identified application and the Bourke-Dunphy et al. (U.S. 6,449,642) with a effective filing date of September 15, 1998 based (hereinafter referred to as "Bourke")..

9) I was hired as a contractor by Brian Styles prior to September 15, 1998 to begin writing software code to implement the above-referenced patent application. A partial listing of software modules and the contents of three source code modules (splashscreen.h; scriptm.cpp; and scriptmdlg.cpp) each with dates of original creation prior to September 15, 1998 are attached hereto. I have diligently worked on this code to carry out this invention from prior to September 15, 1998 through the end of January 21, 1999, after which date the first public release of the code was made. This product has enjoyed great commercial success and has led to the formation of the company ScriptLogic Corporation in Boca Raton, Florida with sales of over 10 million dollars in 2003.

I, the undersigned, hereby declare that all statements made herein are of my own knowledge and are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.



Charles B. Bucklew. June 30, 2004

EXHIBIT FOR AFFIDAVITS

1.KIX
19981230.CSV
19981231.CSV
19990101.CSV
2.KIX
A
A.KIX
A.SCR
ACCACC.CPP
ACCACC.H
ACCOUNT.CPP
ACCOUNT.H
ADF.KIX
Advanced.cpp
Advanced.h
AntiV.cpp
AntiV.h
AppLauncher.cpp
AppLauncher.h
AppLauncherData.cpp
AppLauncherData.h
ASDF.KIX
AssignScript.cpp
AssignScript.h
BitBut.cpp
BitBut.h
BitBut2.cpp
BitBut2.h
CALENDAR.CPP
CALENDAR.H
calendar1.cpp
calendar1.h
CDRAW
CHKLIST.TXT
ColorStaticST.cpp
ColorStaticST.h
COMMON.RTF
commonD.cpp
commonD.h
commonP.cpp
commonP.h
commonS.cpp
commonS.h
CreateMAPI.kix
CUSTOM.KIX

custom_radio_buttons.zip
DACLWRAP.CPP
DACLWRAP.H
DD.KIX
Drives.cpp
Drives.h
DrivesData.cpp
DrivesData.h
DynVarChooser.cpp
DynVarChooser.h
EditCustomScripts.cpp
EditCustomScripts.h
EditMBox.cpp
EditMBox.h
EditorPaths.cpp
EditorPaths.h
EditTargetList.cpp
EditTargetList.h
ENCRYPT.EXE
ENGINE.DLL
EnvData.cpp
EnvData.h
Environ.cpp
Environ.h
FILESEC.CPP
FILESEC.H
FolderBrowse.cpp
FolderBrowse.h
FONT.CPP
FONT.H
FONT1.CPP
FONT1.H
General.cpp
General.h
GroupBrowse.cpp
GroupBrowse.h
HHUPD.EXE
HLP
HLP.H
HTMLHELP.H
HTMLHELP.LIB
Internet.cpp
Internet.h
InternetData.cpp
InternetData.h

ITEK.ICO
ITEKt.ico
LICENSE.TXT
list.txt
ListBoxCtrl.cpp
ListBoxCtrl.h
LOCK.FIL
Logging.cpp
Logging.h
LOGO.BMP
LogView.cpp
LogView.h
LoveBug.kix
MakeHelp.bat
ManagerSplash2601.bmp
MAPI.cpp
MAPI.h
messagebox.cpp
messagebox.h
MessageBoxData.cpp
MessageBoxData.h
MH.BAT
MList.cpp
MList.h
ML_ParseINI.cpp
ML_ParseIni.h
Month11.TXT
MPR.LIB
MSAPPS.CPP
MSAPPS.h
NETAPI32.LIB
NetBrowse.cpp
NetBrowse.h
NTCSFData.cpp
NTCSFData.h
NTDisp.cpp
NTDisp.h
NTLegal.cpp
NTLegal.h
NTPolicy.cpp
NTPolicy.h
NTSCRC.KIX
NTscript Manager Bitmap.bmp
ntscript manager logo 2.bmp
ntscript splash screen logo 2.BMP

ntscript splash screen logo.bmp
NTScript Stuff
NTSMGR.BAK
NTSMGR.BMP
NTsuReset.kix
NTUpdate.cpp
NTUpdate.h
O
OfficeData.cpp
OfficeData.h
OUT.KIX
OXBitmapButton.cpp
OXBitmapButton.h
OXBitmapButton.rc
OXBitmapButtonRes.h
OXDllExt.h
OXSplashWnd.cpp
OXSplashWnd.h
OXSplashWnd.inl
OXSplashWndDIB.cpp
OXSplashWndDIB.h
Page1.cpp
Page1.h
Page2.cpp
Page2.h
ParseINI.cpp
ParseINI.h
PathData.cpp
PathData.h
PickD2.cpp
PickD2.h
PickDate.cpp
PickDate.h
PickPath.cpp
PickPath.h
Policy1.cpp
Policy1.h
Policy1Data.cpp
Policy1Data.h
Policy2.cpp
Policy2.h
POST.KIX
PrinterData.cpp
PrinterData.h
Printers.cpp

```
#if
!defined(AFX_SPLASHSCREEN_H__20856223_48F7_11D2_A4C6_444553540000__INCLUDED_)
#define AFX_SPLASHSCREEN_H__20856223_48F7_11D2_A4C6_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// SplashScreen.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// SplashScreen window

class SplashScreen : public CWnd
{
// Construction
public:
      SplashScreen();

// Attributes
public:

      CBitmap SS;
      CDC   dc;

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(SplashScreen)
      public:
      virtual BOOL Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD
dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID, CCreateContext* pContext
= NULL);
      virtual BOOL DestroyWindow();
      virtual BOOL OnChildNotify(UINT message, WPARAM wParam, LPARAM lParam,
LRESULT* pLResult);
      protected:
      virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~SplashScreen();

      void Go( int );
      void Stop( void );

      // Generated message map functions
protected:
      //{{AFX_MSG(SplashScreen)
      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
      afx_msg void OnPaint();
```

```
	afx_msg void OnGetMinMaxInfo(MINMAXINFO FAR* lpMMI);
	afx_msg BOOL OnNcCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnNcCalcSize(BOOL bCalcValidRects, NCCALCSIZE_PARAMS FAR*
lpncsp);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_SPLASHSCREEN_H__20856223_48F7_11D2_A4C6_444553540000__INCLUDED_)
```

```
// ScriptM.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "ScriptM.h"
#include "ScriptMDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

int nosave;

/////////////////////////////////////////////////////////////////////////////
// CScriptMApp

BEGIN_MESSAGE_MAP(CScriptMApp, CWinApp)
      //{{AFX_MSG_MAP(CScriptMApp)
      //}}AFX_MSG_MAP
      ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CScriptMApp construction

CScriptMApp::CScriptMApp()
{
      // TODO: add construction code here,
      // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CScriptMApp object

CScriptMApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CScriptMApp initialization

CString ScriptPath;
CString LogPath;

extern CString    TextED;
extern CString    ReplED;
CString                 OnSaveCommand = "";

#include <direct.h>
#include "ParseINI.h"

HICON IconList[12];

#include "PickPath.h"
```

```
void TrimString( CString& s )
{
	s.TrimLeft( ' ' );
	s.TrimRight( ' ' );
}

CString NextLine( FILE* f, int len )
{
	CString ret = "";
	char c = '\n';

	while( c == '\n' || c == ' ' )
	c = fgetc( f );

	ret+=c;

	while( c != '\n' && ftell( f ) < len )
	{
		c = fgetc( f );
		if( c != '\n' )
		ret+=c;
	}

	return ret;
}

extern void DoubleAmps( CString& );
extern void LoadTargetList( void );

void GetPaths( void )
{
	ScriptPath = "";
	LogPath = "";
	CurrentDomain = ".";

	char t1[256];
	char temps[256];
	ParseINI pi( true );

	nosave = false;

	static DWORD count;
	static char buf[256];
	static char* b;

	HKEY DisplayKey;
	RegKey = "";
	CompanyName = "";

	bool INIFound;

	if( RegOpenKeyEx( HKEY_CURRENT_USER, "Software\\ScriptLogic", 0,
KEY_ALL_ACCESS, &DisplayKey ) == ERROR_SUCCESS )
```

```
	{
		count = 256;
		sprintf( buf, "" );
		if( RegQueryValueEx( DisplayKey, "RegistrationCode", 0, NULL,
(LPBYTE)buf, &count ) == ERROR_SUCCESS )
			RegKey = buf;

		count = 256;
		sprintf( buf, "" );
		if( RegQueryValueEx( DisplayKey, "RegisteredToComp", 0, NULL,
(LPBYTE)buf, &count ) == ERROR_SUCCESS )
			CompanyName = buf;

		RegCloseKey( DisplayKey );
	}

	if( pi.OpenINI( "slmgr.ini" ) )
	{
		pi.FindValue( "Scripts", t1 );
		ScriptPath = t1;

		DefaultTarget = ScriptPath;

		if( CompanyName == "" )
		{
			pi.FindValue( "Company", t1 );
			CompanyName = t1;
		}

		if( RegKey == "" )
		{
			pi.FindValue( "Code", t1 );
			RegKey = t1;
		}

		pi.FindValue( "Editor", t1 );
		if( strcmp( t1, " " ) == 0 )
		{
			ReplED = "replmgr.exe";
		}
		else
		{
			ReplED = t1;
		}

		pi.FindValue( "ScriptEditor", t1 );
		if( strcmp( t1, " " ) == 0 )
		{
			TextED = "notepad.exe";
		}
		else
		{
			TextED = t1;
		}
```

```
            pi.FindValue( "SMEditor", t1 );
            if( strcmp( t1, " " ) == 0 )
            {
                  SMED = "SLsvcmgr.exe";
            }
            else
            {
                  SMED = t1;
            }

            pi.FindValue( "OnSaveCommand", t1 );
            if( strcmp( t1, " " ) == 0 )
            {
                  OnSaveCommand = "";
            }
            else
            {
                  OnSaveCommand = t1;
            }

            TrimString( ScriptPath );
            TrimString( TextED );
            TrimString( CompanyName );
            TrimString( RegKey );
            TrimString( ReplED );
            TrimString( SMED );

            INIFound = true;
      }
      else
      {
            SS2.ShowWindow( SW_HIDE );
            char temps[1024];
            char cwd[MAX_PATH];

            GetCurrentDirectory(MAX_PATH, cwd );
            sprintf( temps, ResString( IDS_SLMGRINI_NF ), cwd );
            MessageBox( NULL, temps, ResString( IDS_SLMGRINI_NF2 ), MB_OK |
MB_ICONWARNING );
            INIFound = false;
      }

      if( INIFound ) LoadTargetList();

      if( __argc > 1 ) ScriptPath = __argv[1];

      if( ScriptPath[ ScriptPath.GetLength()-1 ] == '\\' ) ScriptPath =
ScriptPath.Left( ScriptPath.GetLength()-1 );

      sprintf( temps, "%s\\%s", (LPCTSTR)ScriptPath, "wtest" );

      FILE* f = NULL;
      int t = clock();
```

```
	while( f == NULL && (clock()-t < (CLOCKS_PER_SEC*5) ))
		f = fopen( temps, "w" );

	int rook = false;

	if( f == NULL )
	{
		SS2.ShowWindow( SW_HIDE );
		MessageBox( NULL, ResString( IDS_PATH_RO ), ResString(
IDS_PATH_ERROR ), MB_OK | MB_ICONWARNING );
		sprintf( t1, ResString( IDS_PATH_RO2 ), (LPCTSTR)ScriptPath );
		rook = true;
	}
	else
	{
		fclose( f );
		DeleteFile( temps );
		rook = (!(GetFileAttributes( (LPCTSTR)ScriptPath ) &
FILE_ATTRIBUTE_DIRECTORY ) || (GetFileAttributes( (LPCTSTR)ScriptPath ) ==
0xFFFFFFFF) || (GetFileAttributes( (LPCTSTR)ScriptPath ) &
FILE_ATTRIBUTE_READONLY));
		sprintf( t1, ResString( IDS_PATH_RO2 ), (LPCTSTR)ScriptPath );
	}

	if( !rook )
	{
		sprintf( temps, "%s\\SLconfig.kix", (LPCTSTR)ScriptPath );
		f = fopen( temps, "r" );

		if( f == NULL )
		{
			rook = true;
			sprintf( t1, ResString( IDS_SCRIPT_NF ), (LPCTSTR)ScriptPath
);
		}
		else
		{
			fclose( f );
		}
	}

	if( rook )
	{
		PickPath* PP;

		PP = new PickPath();

		PP->UseTargetList = INIFound;
		PP->m_Message = t1;
		PP->VerifyReadAccess = true;
		PP->CheckForFile = true;
		PP->FileToCheckFor = "SLconfig.kix";
		PP->FileCheckError = ResString( IDS_SCRIPT_NF2 );
```

```
		SS2.ShowWindow( SW_HIDE );

		if( PP->DoModal() != IDCANCEL )
		{
			ScriptPath = PP->m_Path;

			SS2.ShowWindow( SW_HIDE );

			if( MessageBox( NULL, ResString( IDS_SAVE_INI ), ResString(
IDS_UPDATE_INI ), MB_YESNO ) == IDYES )
			{
				FILE* f = fopen( "slmgr.ini", "r" );
				if( f == NULL )
				{
					f = fopen( "slmgr.ini", "w" );

					fprintf( f, "[File Locations]\n" );
					fprintf( f, "Scripts=%s\n", ScriptPath );

					fclose( f );
				}
				else
				{
					CString Lines[256];
					int l,lines;
					fseek( f, 0, SEEK_END );
					l = ftell( f );
					fseek( f, 0, SEEK_SET );
					lines = 0;
					while( ftell( f ) < l )
					{
						Lines[lines] = NextLine( f, l );
						lines++;
					}
					fclose( f );

					f = fopen( "slmgr.ini", "w" );
					int x;

					for( x=0;x<lines;x++ )
					{
						CString t1;

						if( Lines[x].GetLength() < 7 )
						{
							fprintf( f, "%s\n", Lines[x] );
						}
						else
						{
							t1 = Lines[x].Left(7);
							t1.MakeUpper();

							if( t1 == "SCRIPTS" )
```

```
                                        {
                                              fprintf( f, "Scripts=%s\n",
ScriptPath );
                                        }
                                        else
                                        {
                                              fprintf( f, "%s\n", Lines[x] );
                                        }

                                  }
                            }
                      }

                      fclose( f );
                }
          }
          else
          {
                nosave = true;
                MessageBox( NULL, ResString( IDS_ONLY_LOG ), ResString(
IDS_NO_LOC ), MB_OK );
                ScriptPath = ".";
          }

          delete PP;
    }
}

#include <shlobj.h>

extern HICON PIcon;

BOOL CScriptMApp::InitInstance()
{
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

    SS2.LoadBitmap( MAKEINTRESOURCE( IDB_BITMAP16 ), false );
    SS2.Show( -1, NULL );

    char SysPath[MAX_PATH];
    GetSystemDirectory( SysPath, MAX_PATH );

    int;

    CString sp = SysPath;
    if( sp[sp.GetLength()-1] != '\\' ) sp+='\\';
    CString ocxPath = sp+"hhctrl.ocx";

    bool oldocx = true;

    CFileStatus cfs;
```

```
	if( CFile::GetStatus( ocxPath, cfs ) )
	{
		if( cfs.m_ctime.GetYear() >= 1999 )
		{
			oldocx = false;
		}
	}

	if( oldocx )
	if( GetFileAttributes( "hhupd.exe" ) != 0xFFFFFFFF )
	{
		HKEY K;
		DWORD r2;
		char temps[256];

		int x;

		int splashran = true;

		int dosplash = true;

		RegCreateKeyEx( HKEY_CURRENT_USER,
"Software\\Inteletek\\ScriptLogic\\ManagerRanHere", 0, "REG_SZ",
REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &K, &r2 );

		if( r2 == REG_CREATED_NEW_KEY )
		{
			SS2.ShowWindow( SW_HIDE );

			if( MessageBox( NULL, ResString( IDS_HTMLHELP1 ), ResString(
IDS_UPDATE_SYSTEM ), MB_YESNO ) == IDYES )
			{
				_spawnlp( _P_WAIT, "hhupd.exe", "hhupd.exe", "/Q", NULL
);
				MessageBox( NULL, ResString( IDS_UPDATE_DONE ),
ResString( IDS_UPDATE_COMPLETE ), MB_OK );
				exit(0);
			}
			else
			{
				MessageBox( NULL, ResString( IDS_UPDATE_NOT ),
ResString( IDS_NOT_UPDATED ), MB_OK );
			}
		}

		RegCloseKey( K );
	}

	AfxOleInit();
	AfxEnableControlContainer();

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a
shared DLL
```

```
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	NIcon = LoadIcon( IDI_NETSYM );
	PIcon = LoadIcon( IDI_PENCIL );
	IconList[0] = LoadIcon( IDI_NETWORK );
	IconList[1] = LoadIcon( IDI_DOMAIN );
	IconList[2] = LoadIcon( IDI_SYSTEM );
	IconList[3] = LoadIcon( IDI_FOLDER );
	IconList[4] = LoadIcon( IDI_PRINTER );
	IconList[5] = LoadIcon( IDI_NETWORK );
	IconList[6] = LoadIcon( IDI_GROUP );
	IconList[7] = LoadIcon( IDI_USER );

	char wpath[MAX_PATH];

	GetWindowsDirectory( wpath, MAX_PATH );
	CString duh;
	duh = wpath;
	if( duh.GetLength() > 3 ) duh = duh.Left(3);

	SHFILEINFO sfi;
	memset( &sfi, 0, sizeof( sfi ) );
	CoInitialize( NULL );

	SHGetFileInfo( (LPCTSTR)duh, 0, &sfi, sizeof( SHFILEINFO ), SHGFI_ICON |
SHGFI_SMALLICON );
	IconList[8] = sfi.hIcon;
	SHGetFileInfo( wpath, 0, &sfi, sizeof( SHFILEINFO ), SHGFI_ICON |
SHGFI_SMALLICON );
	IconList[9] = sfi.hIcon;

	IconList[10] = LoadIcon( IDI_GLOBALGROUP );

	GetPaths();

	CScriptMDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
```

}

```
// ScriptMDlg.cpp : implementation file
//

#include "stdafx.h"
#include "ScriptM.h"
#include "ScriptMDlg.h"
#include "LogView.h"
#include "PickDate.h"
#include "PickD2.h"
#include "MList.h"
#include "winnetwk.h"
#include "PW2.h"
#include "ParseINI.h"
#include "TestWnd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern bool ForceDirty;

int Users;
char UserList[MAXUSERS][64];
int Groups;
char GroupList[MAXGROUPS][256];
int Servers;
char ServerList[MAXSERVERS][256];

bool runbenabled = false;

CString CurrentDomain;
CString     ASmgrPath = "";
extern CString OnSaveCommand;

#define EXIT_OK         0
#define EXIT_NOPUB      1
#define EXIT_NOSAVE     2

int ExitState = EXIT_OK;

bool Enumeration = true;

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
      CAboutDlg();

// Dialog Data
      //{{AFX_DATA(CAboutDlg)
```

```
	enum { IDD = IDD_ABOUTBOX };
	CString	m_VerInfo;
	CString	m_Company;
	CString	m_Key;
	CString	m_UserInfo;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	public:
	virtual void WinHelp(DWORD dwData, UINT nCmd = HELP_CONTEXT);
	virtual BOOL DestroyWindow();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	afx_msg BOOL OnHelpInfo(HELPINFO* pHelpInfo);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	m_VerInfo = _T("");
	m_Company = _T("");
	m_Key = _T("");
	m_UserInfo = _T("");
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	DDX_Text(pDX, IDC_STATICTEXT, m_VerInfo);
	DDX_Text(pDX, IDC_STATICTEXT2, m_Company);
	DDX_Text(pDX, IDC_STATICTEXT3, m_Key);
	DDX_Text(pDX, IDC_STATICTEXT5, m_UserInfo);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
	ON_WM_HELPINFO()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CScriptMDlg dialog
```

```
CScriptMDlg::CScriptMDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CScriptMDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CScriptMDlg)
	m_Bottom = _T("");
	m_Location = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CScriptMDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CScriptMDlg)
	DDX_Control(pDX, IDC_BUTTON2, m_EB);
	DDX_Control(pDX, IDC_SAVESCRIPT, m_SS);
	DDX_Control(pDX, IDC_EDITREPLBAT2, m_ASmgr);
	DDX_Control(pDX, IDC_COMBO2, m_LocationC);
	DDX_Control(pDX, IDC_STARTUPBAT, m_SUB);
	DDX_Control(pDX, IDC_EDITREPLBAT, m_editrb);
	DDX_Control(pDX, IDC_BUTTON1, m_b1);
	DDX_Control(pDX, IDC_CUSTOMSCRIPT, m_cs);
	DDX_Control(pDX, IDC_REPLBATCH, m_runrb);
	DDX_Text(pDX, IDC_BOTTOM, m_Bottom);
	DDX_CBString(pDX, IDC_COMBO2, m_Location);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CScriptMDlg, CDialog)
	//{{AFX_MSG_MAP(CScriptMDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
	ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
	ON_BN_CLICKED(IDC_SAVESCRIPT, OnSavescript)
	ON_WM_CLOSE()
	ON_BN_CLICKED(IDC_REVIEWLOG, OnReviewlog)
	ON_BN_CLICKED(IDC_CUSTOMSCRIPT, OnCustomscript)
	ON_BN_CLICKED(IDC_STARTUPBAT, OnStartupbat)
	ON_BN_CLICKED(IDC_REPLBATCH, OnReplbatch)
	ON_BN_CLICKED(IDC_EDITREPLBAT, OnEditreplbat)
	ON_BN_CLICKED(IDC_BITBUT, OnBitbut)
	ON_BN_CLICKED(IDB_BITMAP2, OnBitmap2)
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_BUTTON3, OnTestBrowse)
	ON_BN_CLICKED(IDC_HELP, OnHelp)
	ON_COMMAND(ID_FILE_SAVE, OnFileSave)
	ON_COMMAND(ID_HELP_SCRIPTLOGICONTHEWEB, OnHelpScriptlogicontheweb)
	ON_COMMAND(ID_HELP_CONTENTS, OnHelpContents)
	ON_COMMAND(ID_EDIT_CUSTOMSCRIPT, OnEditCustomscript)
	ON_COMMAND(ID_EDIT_CUSTOMSCRIPT2, OnEditCustomscript2)
	ON_COMMAND(ID_EDIT_REPLICATIONBATCH, OnEditReplicationbatch)
```

```
      ON_COMMAND(ID_EDIT_SCRIPTLOGICCONFIGURATION,
OnEditScriptlogicconfiguration)
      ON_COMMAND(ID_FILE_EXIT, OnFileExit)
      ON_COMMAND(ID_FILE_REPLICATE, OnFileReplicate)
      ON_COMMAND(ID_VIEW_LOGS, OnViewLogs)
      ON_COMMAND(ID_HELP_ABOUTSCRIPTLOGIC, OnHelpAboutscriptlogic)
      ON_COMMAND(ID_HELP_REGISTER, OnHelpRegister)
      ON_COMMAND(ID_EDIT_VALIDATIONDEFAULTS, OnEditValidationdefaults)
      ON_WM_HELPINFO()
      ON_BN_CLICKED(IDC_ADDLOCATION, OnAddlocation)
      ON_CBN_SELCHANGE(IDC_COMBO2, OnSelchangeCombo2)
      ON_CBN_DROPDOWN(IDC_COMBO2, OnDropdownCombo2)
      ON_COMMAND(ID_TOOLS_EDITSCRIPTLIST, OnToolsEditscriptlist)
      ON_BN_CLICKED(IDC_EDITREPLBAT2, OnASmgr)
      ON_COMMAND(ID_TOOLS_AUTOSHAREMANAGER, OnToolsAutosharemanager)
      ON_COMMAND(ID_OPTIONS_REPLMANAGERPATH, OnOptionsReplmanagerpath)
      ON_UPDATE_COMMAND_UI(ID_FILE_SAVE, OnUpdateFileSave)
      ON_BN_CLICKED(IDC_EDITREPLBAT3, OnServiceManager)
      ON_COMMAND(ID_TOOLS_SERVICEMANAGER, OnToolsServicemanager)
      ON_COMMAND(ID_TOOLS_ASSIGNSCRIPT, OnToolsAssignscript)
      ON_COMMAND(ID_TOOLS_SYSTEMOPTIONS, OnToolsSystemoptions)
      ON_COMMAND(ID_TOOLS_CUSTOMSCRIPTMANAGER, OnToolsCustomscriptmanager)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CScriptMDlg message handlers

#define MAXRESOURCES 1500

void DisplayStruct( LPNETRESOURCE n )
{
      char temps[2048];

      char s1[256];
      char s2[256];
      char s3[256];
      char s4[256];

      if( n->dwScope == RESOURCE_CONNECTED ) sprintf( s1, "RESOURCE_CONNECTED"
);
      if( n->dwScope == RESOURCE_GLOBALNET ) sprintf( s1, "RESOURCE_GLOBALNET"
);
      if( n->dwScope == RESOURCE_REMEMBERED ) sprintf( s1,
"RESOURCE_REMEMBERED" );

      if( n->dwType == RESOURCETYPE_ANY ) sprintf( s2, "RESOURCETYPE_ANY" );
      if( n->dwType == RESOURCETYPE_DISK ) sprintf( s2, "RESOURCETYPE_DISK" );
      if( n->dwType == RESOURCETYPE_PRINT ) sprintf( s2, "RESOURCETYPE_PRINT"
);

      if( n->dwDisplayType == RESOURCEDISPLAYTYPE_DOMAIN ) sprintf( s3,
"RESOURCEDISPLAYTYPE_DOMAIN" );
```

```
      if( n->dwDisplayType == RESOURCEDISPLAYTYPE_SERVER ) sprintf( s3,
"RESOURCEDISPLAYTYPE_SERVER" );
      if( n->dwDisplayType == RESOURCEDISPLAYTYPE_SHARE ) sprintf( s3,
"RESOURCEDISPLAYTYPE_SHARE" );
      if( n->dwDisplayType == RESOURCEDISPLAYTYPE_GENERIC ) sprintf( s3,
"RESOURCEDISPLAYTYPE_GENERIC" );

      if( n->dwUsage == RESOURCEUSAGE_CONNECTABLE ) sprintf( s4,
"RESOURCEUSAGE_CONNECTABLE" );
      if( n->dwUsage == RESOURCEUSAGE_CONTAINER ) sprintf( s4,
"RESOURCEUSAGE_CONTAINER" );

      sprintf( temps, "scp:%s,typ:%s,dt:%s,usg:%s,rn:%s,ln:%s,cmt:%s,pvdr:%s",
s1,s2,s3,s4, n->lpLocalName, n->lpRemoteName, n->lpComment, n->lpProvider );

      MessageBox( NULL, temps, temps, MB_OK );

}

void Chop1( char* in, char* out )
{
      int x,y;

      for( x=0,y=2;y<strlen( in );x++,y++ )
      {
            out[x] = in[y];
      }

      out[x] = 0;
}

void Chop2( char* in, char* out )
{
       int x,y;
       int c = 0;
       CString t;

       // This is dumb looking, but some compiler bugs are just to BAD

       char ti[256];

       sprintf( ti, "%s", in );

       t = ti;

     c = t.ReverseFind( '\\' );
       sprintf( out, "%s", (LPCTSTR)(t.Right( t.GetLength() - c - 1)) );
       return;

       while( c < 3 && x < strlen( in ))
       {
            MessageBox( NULL, (LPCTSTR)t, in, MB_OK );

            if( t[x] == '\\' )
```

```
            {
                  MessageBox( NULL, "...", "...", MB_OK );
                  c++;
            }
            x++;
       }

       for( y=0;x<strlen( in );x++,y++ )
       {
            out[y] = in[x];
       }

       out[y] = 0;

       MessageBox( NULL, out, in, MB_OK );
}

BOOL EnumerateServersFrom( LPNETRESOURCE source )
{
    DWORD dwResult, dwResultEnum;     HANDLE hEnum;
    DWORD cbBuffer = 16384;      // 16K is a good size
    DWORD cEntries = 0xFFFFFFFF; // enumerate all possible entries
    LPNETRESOURCE lpnrLocal;     // pointer to enumerated structures
    DWORD i;
      char ts[256];

      dwResult = WNetOpenEnum(RESOURCE_GLOBALNET,
                                        RESOURCETYPE_ANY,         0,
// enumerate all resources
                                          source,               // NULL first
time this function is called
                                          &hEnum);            // handle to
resource

    if (dwResult != NO_ERROR)
      {
        return false;
      }

      do
      {
        // Allocate memory for NETRESOURCE structures.
        lpnrLocal = (LPNETRESOURCE) GlobalAlloc(GPTR, cbBuffer);

        dwResultEnum = WNetEnumResource(hEnum, // resource handle
            &cEntries,               // defined locally as 0xFFFFFFFF
            lpnrLocal,               // LPNETRESOURCE
            &cbBuffer);              // buffer size

            if (dwResultEnum == NO_ERROR)
            {
            for(i = 0; i < cEntries; i++)
                  {
                        Chop1( lpnrLocal[i].lpRemoteName, ts );
```

```
                        sprintf( ServerList[ Servers ], "%s", ts );
                        Servers++;
                    }
                }
                else if (dwResultEnum != ERROR_NO_MORE_ITEMS)
                {
                        break;
                }

        } while(dwResultEnum != ERROR_NO_MORE_ITEMS);

        GlobalFree((HGLOBAL) lpnrLocal);
        WNetCloseEnum(hEnum);

        return false;
}

BOOL EnumerateSharesFrom( LPNETRESOURCE source, CComboBox& b )
{
    DWORD dwResult, dwResultEnum;     HANDLE hEnum;
    DWORD cbBuffer = 16384;      // 16K is a good size
    DWORD cEntries = 0xFFFFFFFF; // enumerate all possible entries
    LPNETRESOURCE lpnrLocal;     // pointer to enumerated structures
    DWORD i;
      char ts1[256];
      char temps[256];

      while( b.DeleteString( 0 ) != CB_ERR );

      dwResult = WNetOpenEnum(RESOURCE_GLOBALNET,
                                        RESOURCETYPE_ANY,         0,
// enumerate all resources
                                          source,               // NULL first
time this function is called
                                          &hEnum);            // handle to
resource

    if (dwResult != NO_ERROR)
      {
        return false;
      }

      do
      {
        // Allocate memory for NETRESOURCE structures.
        lpnrLocal = (LPNETRESOURCE) GlobalAlloc(GPTR, cbBuffer);

        dwResultEnum = WNetEnumResource(hEnum, // resource handle
            &cEntries,               // defined locally as 0xFFFFFFFF
            lpnrLocal,               // LPNETRESOURCE
            &cbBuffer);              // buffer size

            if (dwResultEnum == NO_ERROR)
            {
```

```
            for(i = 0; i < cEntries; i++)
                  {
                        sprintf( ts1, "%s", lpnrLocal[i].lpRemoteName );
                        Chop2( ts1, temps );
                        b.AddString( temps );
                  }
            }
            else if (dwResultEnum != ERROR_NO_MORE_ITEMS)
            {
                  break;
            }

      } while(dwResultEnum != ERROR_NO_MORE_ITEMS);

      GlobalFree((HGLOBAL) lpnrLocal);
      WNetCloseEnum(hEnum);

      return false;
}

BOOL EnumeratePrintersFrom( LPNETRESOURCE source, CComboBox& b )
{
    DWORD dwResult, dwResultEnum;     HANDLE hEnum;
    DWORD cbBuffer = 16384;      // 16K is a good size
    DWORD cEntries = 0xFFFFFFFF; // enumerate all possible entries
    LPNETRESOURCE lpnrLocal;     // pointer to enumerated structures
    DWORD i;
      char ts1[256];
      char temps[256];

      while( b.DeleteString( 0 ) != CB_ERR );

      dwResult = WNetOpenEnum(RESOURCE_GLOBALNET,
                                    RESOURCETYPE_PRINT,         0,
// enumerate all resources
                                      source,               // NULL first
time this function is called
                                      &hEnum);            // handle to
resource

    if (dwResult != NO_ERROR)
      {
        return false;
      }

      do
      {
        // Allocate memory for NETRESOURCE structures.
        lpnrLocal = (LPNETRESOURCE) GlobalAlloc(GPTR, cbBuffer);

        dwResultEnum = WNetEnumResource(hEnum, // resource handle
            &cEntries,               // defined locally as 0xFFFFFFFF
            lpnrLocal,               // LPNETRESOURCE
            &cbBuffer);              // buffer size
```

```
            if (dwResultEnum == NO_ERROR)
            {
            for(i = 0; i < cEntries; i++)
                  {
                        sprintf( ts1, "%s", lpnrLocal[i].lpRemoteName );
                        Chop2( ts1, temps );
                        b.AddString( temps );
                  }
            }
            else if (dwResultEnum != ERROR_NO_MORE_ITEMS)
            {
                     break;
            }

      } while(dwResultEnum != ERROR_NO_MORE_ITEMS);

      GlobalFree((HGLOBAL) lpnrLocal);
      WNetCloseEnum(hEnum);

      return false;
}

BOOL FindIn( const char* name, LPNETRESOURCE source, LPNETRESOURCE result, int
r = 1 )
{
    DWORD dwResult, dwResultEnum;     HANDLE hEnum;
    DWORD cbBuffer = 16384;      // 16K is a good size
    DWORD cEntries = 0xFFFFFFFF; // enumerate all possible entries
    LPNETRESOURCE lpnrLocal;     // pointer to enumerated structures
    DWORD i;

      r--;

      char temps[256];

      CString n1,n2;

      n1 = name;
      n1.MakeUpper();

      if( r < 0 ) return false;

      dwResult = WNetOpenEnum(RESOURCE_GLOBALNET,
                                          RESOURCETYPE_ANY,          0,
// enumerate all resources
                                            source,               // NULL first
time this function is called
                                            &hEnum);            // handle to
resource

    if (dwResult != NO_ERROR)
      {
        return false;
```

```
        }

        do
        {
          // Allocate memory for NETRESOURCE structures.
          lpnrLocal = (LPNETRESOURCE) GlobalAlloc(GPTR, cbBuffer);

          dwResultEnum = WNetEnumResource(hEnum, // resource handle
              &cEntries,               // defined locally as 0xFFFFFFFF
              lpnrLocal,               // LPNETRESOURCE
              &cbBuffer);              // buffer size

              if (dwResultEnum == NO_ERROR)
              {
              for(i = 0; i < cEntries; i++)
                  {
                        n2 = lpnrLocal[i].lpRemoteName;

                        n2.MakeUpper();

//                      if( strcmp( lpnrLocal[i].lpRemoteName, name ) == 0 )

                        if( strcmp( (LPCTSTR)n1, (LPCTSTR)n2 ) == 0 )
                        {
                              memcpy( (void*)result, &lpnrLocal[i], sizeof(
NETRESOURCE ) );
                              return true;
                        }

                        // If this NETRESOURCE is a container, call the
function
                // recursively.
                        if(RESOURCEUSAGE_CONTAINER ==
                        (lpnrLocal[i].dwUsage & RESOURCEUSAGE_CONTAINER))
                    if( FindIn( name, &lpnrLocal[i], result, r ) )
                              {
                                    GlobalFree((HGLOBAL) lpnrLocal);
                                    WNetCloseEnum(hEnum);
                                    return true;
                              }
                  }
              }
              else if (dwResultEnum != ERROR_NO_MORE_ITEMS)
              {
                        break;
              }

        } while(dwResultEnum != ERROR_NO_MORE_ITEMS);

        GlobalFree((HGLOBAL) lpnrLocal);
        WNetCloseEnum(hEnum);

        return false;
}
```

```
BOOL WINAPI EnumerateFunc( LPNETRESOURCE lpnr)
{
    DWORD dwResult, dwResultEnum;     HANDLE hEnum;
    DWORD cbBuffer = 16384;      // 16K is a good size
    DWORD cEntries = 0xFFFFFFFF; // enumerate all possible entries
    LPNETRESOURCE lpnrLocal;     // pointer to enumerated structures
    DWORD i;      dwResult = WNetOpenEnum(RESOURCE_GLOBALNET,
        RESOURCETYPE_ANY,         0,                 // enumerate all resources
        lpnr,               // NULL first time this function is called
        &hEnum);            // handle to resource
    if (dwResult != NO_ERROR)
      {

        // An application-defined error handler is demonstrated in the
        // section titled "Retrieving Network Errors."
        return FALSE;     }      do {
        // Allocate memory for NETRESOURCE structures.
        lpnrLocal = (LPNETRESOURCE) GlobalAlloc(GPTR, cbBuffer);
        dwResultEnum = WNetEnumResource(hEnum, // resource handle
            &cEntries,              // defined locally as 0xFFFFFFFF
            lpnrLocal,              // LPNETRESOURCE
            &cbBuffer);             // buffer size
        if (dwResultEnum == NO_ERROR) {
            for(i = 0; i < cEntries; i++)             {
                // Following is an application-defined function for
                // displaying contents of NETRESOURCE structures.
                DisplayStruct(&lpnrLocal[i]);                  //
                // If this NETRESOURCE is a container, call the function
                // recursively.
                        if(RESOURCEUSAGE_CONTAINER ==
                        (lpnrLocal[i].dwUsage & RESOURCEUSAGE_CONTAINER))
                    if(!EnumerateFunc(&lpnrLocal[i])) ;
            }         }
        else if (dwResultEnum != ERROR_NO_MORE_ITEMS) {
            break;          }     }
    while(dwResultEnum != ERROR_NO_MORE_ITEMS);
    GlobalFree((HGLOBAL) lpnrLocal);      dwResult = WNetCloseEnum(hEnum);
    if(dwResult != NO_ERROR) {
        return FALSE;     }      return TRUE;
}

void EnumShares( const char* s, CComboBox& b )
{
/*
      char ts[256];

      sprintf( ts, "\\\\%s", s );

#ifdef C4_WIN_NT
      NETRESOURCE nr;
      NETRESOURCE nr2;

      if( FindIn( (LPCTSTR)CurrentDomain, NULL, &nr2, 2 ) )
```

```
	{
		if( FindIn( ts, &nr2, &nr, 1 ) )
		{
			EnumerateSharesFrom( &nr, b );
		}
		else
		{
			if( FindIn( ts, &nr2, &nr, 2 ) )
			{
				EnumerateSharesFrom( &nr, b );
			}
			else
			{
				if( FindIn( ts, &nr2, &nr, 3 ) )
				{
					EnumerateSharesFrom( &nr, b );
				}
			else
				{
					MessageBox( NULL, "Server Not Found",
"Network Error", MB_OK );
					return;
				}
			}
		}
	}
	else
	{
		MessageBox( NULL, "Local Domain Not Found", "Network Error", MB_OK
);
		return;
	}

#endif

#ifdef C4_WIN_NT
	PW2* PW;
	PW = new PW2();
	PW->Create( IDD_PLEASEWAIT2 );

	SetCursor(LoadCursor(NULL, IDC_WAIT));

	while( b.DeleteString( 0 ) != CB_ERR );

	HANDLE eh;

	NETRESOURCE nr;
	NETRESOURCE rlist[MAXRESOURCES];
	DWORD size = (sizeof( NETRESOURCE ) * MAXRESOURCES);
	DWORD count = MAXRESOURCES;
	SERVER_INFO_100* i;

	wchar_t wcs[256];
	mbstowcs( wcs, s, 256 );
```

```
	if( s[0] == 0 )
	{
		SetCursor(LoadCursor(NULL, IDC_ARROW));
		PW->DestroyWindow();
		b.AddString( "<Invalid Server>" );
		return;
	}

	if( NetServerGetInfo( (char*)wcs, 100, (BYTE**)&i ) != NERR_Success )
	{
		SetCursor(LoadCursor(NULL, IDC_ARROW));
		PW->DestroyWindow();
		b.AddString( "<Invalid Server>" );
		MessageBox( NULL, "Server Not Found", "Share Enum Error", MB_OK |
MB_ICONWARNING );
		return;
	}

	NetApiBufferFree( i );

	char temps[256];
	sprintf( temps, "\\\\%s", s );

	nr.lpRemoteName = (char*)temps;
	nr.dwScope = RESOURCE_GLOBALNET;
	nr.dwType = RESOURCETYPE_ANY;
	nr.dwDisplayType = RESOURCEDISPLAYTYPE_SERVER;
	nr.dwUsage = RESOURCEUSAGE_CONTAINER;

	DWORD ret;

	ret = WNetOpenEnum( RESOURCE_GLOBALNET, RESOURCETYPE_ANY, 0, &nr, &eh );

	if( ret == ERROR_NOT_CONTAINER ) { MessageBox( NULL, "Not Container",
"Open Error", MB_OK ); return; };
	if( ret == ERROR_INVALID_PARAMETER ) { MessageBox( NULL, "Invalid
Paramater", "Open Error", MB_OK ); return; };
	if( ret == ERROR_NO_NETWORK ) { MessageBox( NULL, "No Network", "Open
Error", MB_OK ); return; };
	if( ret == ERROR_EXTENDED_ERROR ) { MessageBox( NULL, "Extended Error",
"Open Error", MB_OK ); return; };

	if( ret != NO_ERROR )
	{
		ret = FormatMessage(	FORMAT_MESSAGE_FROM_SYSTEM,
					0,
					GetLastError(),
					0,
					temps,
					256, 0 );


		MessageBox( NULL, temps, "Open UFO Error", MB_OK );
```

```
            return;
      };

      ret = WNetEnumResource( eh, &count, (LPVOID)&rlist, &size );

      if( ret == ERROR_MORE_DATA ) { MessageBox( NULL, "More Data", "Error",
MB_OK ); return; };
      if( ret == ERROR_INVALID_HANDLE ) { MessageBox( NULL, "Invalid Handle",
"Error", MB_OK ); return; };
      if( ret == ERROR_NO_NETWORK ) { MessageBox( NULL, "No Network","Error",
MB_OK ); return; };
      if( ret == ERROR_EXTENDED_ERROR ) { MessageBox( NULL, "Extended
Error","Error", MB_OK ); return; };

      int x;

      //sprintf( temps, "Count:%i", count );
      //MessageBox( NULL, temps, temps, MB_OK );

      sprintf( temps, "Count:%i", count );
      MessageBox( NULL, temps, temps, MB_OK );

      for( x=0;x<count;x++ )
      {
            char chopchop[1024];
            int c = 0;
            int y = 0;
            int z = 0;

            while( c < 3 && y < strlen( rlist[x].lpRemoteName ) )
            {
                  if( rlist[x].lpRemoteName[y] == '\\' ) c++;
                  y++;
            }

            for( ;y<strlen( rlist[x].lpRemoteName );y++ )
            {
                  chopchop[z] = rlist[x].lpRemoteName[y];
                  z++;
            }

            chopchop[z] = 0;
            b.AddString( chopchop );
      }

      WNetCloseEnum( eh );

      PW->DestroyWindow();
      SetCursor(LoadCursor(NULL, IDC_ARROW));
#endif
*/
}

void EnumPrinters( const char* s, CComboBox& b )
```

```
{
/*
      NETRESOURCE nr;
      NETRESOURCE nr2;

      char ts[256];

      sprintf( ts, "\\\\%s", s );

#ifdef C4_WIN_NT
      if( FindIn( (LPCTSTR)CurrentDomain, NULL, &nr2, 2 ) )
      {


            if( FindIn( ts, &nr2, &nr, 1 ) )
            {
                  EnumeratePrintersFrom( &nr, b );
            }
            else
            {
                  if( FindIn( ts, &nr2, &nr, 2 ) )
                  {
                        EnumeratePrintersFrom( &nr, b );
                  }
                  else
                  {
                        if( FindIn( ts, &nr2, &nr, 3 ) )
                        {
                              EnumeratePrintersFrom( &nr, b );
                        }
                  else
                        {
                                    MessageBox( NULL, "Server Not Found",
"Network Error", MB_OK );
                                    return;
                        }
                  }
            }
      }
      else
      {
            MessageBox( NULL, "Local Domain Not Found", "Network Error", MB_OK
);
            return;
      }

#endif

#ifdef C4_WIN_NT
      PW2* PW;
      PW = new PW2();
      PW->Create( IDD_PLEASEWAIT2 );

      SetCursor(LoadCursor(NULL, IDC_WAIT));
```

```
	while( b.DeleteString( 0 ) != CB_ERR );

	HANDLE eh;

	NETRESOURCE nr;
	NETRESOURCE rlist[MAXRESOURCES];
	DWORD size = (sizeof( NETRESOURCE ) * MAXRESOURCES);
	DWORD count = MAXRESOURCES;
	SERVER_INFO_100* i;

	wchar_t wcs[256];
	mbstowcs( wcs, s, 256 );

	if( s[0] == 0 )
	{
		SetCursor(LoadCursor(NULL, IDC_ARROW));
		PW->DestroyWindow();
		b.AddString( "<Invalid Server>" );
		return;
	}

	if( NetServerGetInfo( (char*)wcs, 100, (BYTE**)&i ) != NERR_Success )
	{
		SetCursor(LoadCursor(NULL, IDC_ARROW));
		PW->DestroyWindow();
		b.AddString( "<Invalid Server>" );
		MessageBox( NULL, "Server Not Found", "Share Enum Error", MB_OK |
MB_ICONWARNING );
		return;
	}

	NetApiBufferFree( i );

	char temps[256];
	sprintf( temps, "\\\\%s", s );

	nr.lpRemoteName = (char*)temps;
	nr.dwScope = RESOURCE_GLOBALNET;
	nr.dwType = RESOURCETYPE_ANY;
	nr.dwDisplayType = RESOURCEDISPLAYTYPE_SERVER;
	nr.dwUsage = RESOURCEUSAGE_CONTAINER;

	WNetOpenEnum( RESOURCE_GLOBALNET, RESOURCETYPE_PRINT, 0, &nr, &eh );

	WNetEnumResource( eh, &count, (LPVOID)&rlist, &size );

	int x;

	for( x=0;x<count;x++ )
	{
		char chopchop[256];
		int c = 0;
		int y = 0;
```

```
            int z = 0;

            while( c < 3 && y < strlen( rlist[x].lpRemoteName ) )
            {
                  if( rlist[x].lpRemoteName[y] == '\\' ) c++;
                  y++;
            }

            for( ;y<strlen( rlist[x].lpRemoteName );y++ )
            {
                  chopchop[z] = rlist[x].lpRemoteName[y];
                  z++;
            }

            chopchop[z] = 0;
            b.AddString( chopchop );
      }

      WNetCloseEnum( eh );

      PW->DestroyWindow();
      SetCursor(LoadCursor(NULL, IDC_ARROW));
#endif
*/
}

void GetServers( void )
{
/*
      NETRESOURCE nr;
      NETRESOURCE nr2;

      if( FindIn( (LPCTSTR)CurrentDomain, NULL, &nr, 2 ) )
      {
            EnumerateServersFrom( &nr );
      }
      else
      {
            MessageBox( NULL, "Cannot Locate Domain", "Network Error", MB_OK );
      }

#ifdef C4_WIN_NT
//    PW2* PW;
//    PW = new PW2();
//    PW->Create( IDD_PLEASEWAIT2 );

      //    MessageBox( NULL, "Enumerating Servers", "Enum", MB_OK );

//    EnumerateFunc( NULL );

      SetCursor(LoadCursor(NULL, IDC_WAIT));

      HANDLE eh;
```

```
	NETRESOURCE nr;
	NETRESOURCE rlist[MAXRESOURCES];
	DWORD size = (sizeof( NETRESOURCE ) * MAXRESOURCES);
	DWORD count = MAXRESOURCES;
	SERVER_INFO_100* i;

	char s[256];

	sprintf( s, "%s", (LPCTSTR)CurrentDomain );

	wchar_t wcs[256];
	mbstowcs( wcs, s, 256 );

	if( s[0] == 0 )
	{
		SetCursor(LoadCursor(NULL, IDC_ARROW));
//		PW->DestroyWindow();
//		b.AddString( "<Invalid Server>" );
		return;
	}

//	if( NetServerGetInfo( (char*)wcs, 100, (BYTE**)&i ) != NERR_Success )
//	{
//		SetCursor(LoadCursor(NULL, IDC_ARROW));
//		PW->DestroyWindow();
//		b.AddString( "<Invalid Server>" );
//		MessageBox( NULL, "Server Not Found", "Share Enum Error", MB_OK |
MB_ICONWARNING );
//		return;
//	}

//	NetApiBufferFree( i );

	char temps[256];
	sprintf( temps, "%s", s );

	nr.lpRemoteName = temps;
	nr.dwScope = RESOURCE_GLOBALNET;
	nr.dwType = RESOURCETYPE_ANY;
	nr.dwDisplayType = RESOURCEDISPLAYTYPE_DOMAIN;
	nr.dwUsage = RESOURCEUSAGE_CONTAINER;

	WNetOpenEnum( RESOURCE_GLOBALNET, RESOURCETYPE_ANY, 0, &nr, &eh );

	WNetEnumResource( eh, &count, (LPVOID)&rlist, &size );

	int x;

	Servers = 0;

	for( x=0;x<count;x++ )
	{
		char chopchop[256];
		int c = 0;
```

```
            int y = 0;
            int z = 0;

            char rn[256];

            sprintf( rn, "%s", rlist[x].lpRemoteName );

            while( c < 3 && y < strlen( rn ) )
            {
                  if( rn[c] == '\\' ) c++;
                  y++;
            }

            for( ;y<strlen( rn );y++ )
            {
                  chopchop[z] = rn[y];
                  z++;
            }

            chopchop[z] = 0;

            CString tcs = rn;
            sprintf( chopchop, "%s", (LPCTSTR)(tcs.Right( tcs.GetLength()-2 ))
);

            sprintf( ServerList[Servers], "%s", chopchop );
            Servers++;
      }

      WNetCloseEnum( eh );

//    PW->DestroyWindow();
      SetCursor(LoadCursor(NULL, IDC_ARROW));
#endif

/*
#ifdef C4_WIN_NT
            _SERVER_INFO_100 *buf,*cur;

            DWORD read,total,resumeh,rc,i;
            DWORD prefmaxlen = 512;

            resumeh = 0;

            Servers = 0;

            do
            {
                  buf = NULL;

                  rc = NetServerEnum( NULL, 100, (BYTE**)&buf, prefmaxlen,
&read, &total, SV_TYPE_ALL, NULL, NULL );

                  if( rc != ERROR_MORE_DATA && rc != ERROR_SUCCESS ) break;
```

```
                for( i=0,cur = buf;i<read;++i,++cur )
                {
                        wcstombs( ServerList[Servers], (const wchar_t*)(cur-
>sv100_name), 256 );

                        if( Servers < MAXSERVERS-2 )
                        Servers++;
                }

                if( buf != NULL ) NetApiBufferFree( buf );

        } while( rc == ERROR_MORE_DATA );
#endif
*/
}

extern CString LogonServer;

void GetUsers( void )
{
#ifdef C4_WIN_NT
        LPUSER_INFO_1     pBuf;
        DWORD             numUsers;
        DWORD             could;
        DWORD             resume=0;
        BYTE* server = NULL;
        NetGetDCName( NULL, NULL, &server );
        WCHAR*      logs[1024];

        char cname[256];
        GetEnvironmentVariable( "LogonServer", cname, 256 );
        LogonServer = cname;

        if( LogonServer != "" )
        {
                mbstowcs( (unsigned short*)logs, (const char*)(LPCTSTR)LogonServer,
LogonServer.GetLength()+1 );
        }
        else
        {
                wcscpy( (unsigned short*)logs, (const unsigned short*)server );
        }

        NetUserEnum( (const wchar_t*)logs, 1, 0, (LPBYTE*)&pBuf, 0xFFFFFFFF,
&numUsers, &could, &resume );
        NetApiBufferFree( server );

        Users = 0;

        for( int u=0;u<numUsers;u++ )
        {
                char uname[MAX_PATH];
```

```
            if( (pBuf[u].usri1_flags & UF_NORMAL_ACCOUNT) ||
(pBuf[u].usri1_flags & UF_TEMP_DUPLICATE_ACCOUNT) )
            {
                  wcstombs( UserList[Users], (const
wchar_t*)(pBuf[u].usri1_name), wcslen( pBuf[u].usri1_name ) + 1 );

                  if( Users < MAXUSERS-2 )
                  Users++;
            }
      }

      NetApiBufferFree( pBuf );
#endif
}

void GetLocalGroups( void )
{

#ifdef C4_WIN_NT
            LOCALGROUP_INFO_1 *buf,*cur;
            BYTE *server=NULL;

            DWORD read,total,resumeh,rc,i;
            DWORD prefmaxlen = 512;

            NetGetDCName( NULL, NULL, &server );

            resumeh = 0;

            do
            {
                  buf = NULL;

                  rc = NetLocalGroupEnum( (const unsigned short*)server, 1,
(BYTE**)&buf, prefmaxlen, &read, &total, &resumeh );

                  if( rc != ERROR_MORE_DATA && rc != ERROR_SUCCESS ) break;

                  for( i=0,cur = buf;i<read;++i,++cur )
                  {
                        wcstombs( GroupList[Groups], (const wchar_t*)(cur-
>lgrpi1_name), 256 );

                        if( Groups < MAXGROUPS-2 )
                        Groups++;
                  }

                  if( buf != NULL ) NetApiBufferFree( buf );

            } while( rc == ERROR_MORE_DATA );

            NetApiBufferFree( server );
#endif
}
```

```
void GetGlobalGroups( void )
{
#ifdef C4_WIN_NT
            void *buf;
        NET_DISPLAY_USER *ndu;
        NET_DISPLAY_MACHINE *ndm;
        NET_DISPLAY_GROUP *ndg;
        DWORD read, next_index, rc, i;

            BYTE *server=NULL;

            int level = 3;

            NetGetDCName( NULL, NULL, &server );

        next_index = 0;
        do
        {
                buf = NULL;

                        rc = NetQueryDisplayInformation( (const unsigned
short*)server, level, next_index, 10, 1024, &read, &buf );
                if ( rc != ERROR_MORE_DATA && rc != ERROR_SUCCESS )
                        break;

                        for ( i = 0, ndg = (NET_DISPLAY_GROUP *) buf; i < read;
++ i, ++ ndg )
                        {
                              wcstombs( GroupList[Groups], (const
wchar_t*)(ndg->grpi3_name), 256 );

                              if( Groups < MAXGROUPS-2 )
                              Groups++;
                        }

                        if ( read > 0 )
                              next_index = ((NET_DISPLAY_GROUP *)buf)[read -
1].grpi3_next_index;

                        if ( buf != NULL )
                              NetApiBufferFree( buf );

        } while ( rc == ERROR_MORE_DATA );

            NetApiBufferFree( server );
#endif
}

void CScriptMDlg::EnumerateLocalGroups( void )
{
      Groups = 0;

      if( Enumeration )
```

```
	{

	#ifdef C4_WIN_NT
		SS2.PrintMessage( "Enumerating Local Groups..." );
		GetLocalGroups();

		SS2.PrintMessage( "Enumerating Global Groups..." );
		GetGlobalGroups();

		SS2.PrintMessage( "Enumerating Servers..." );
		GetServers();

		SS2.PrintMessage( "Enumerating Users..." );
		GetUsers();
	#endif

	}

}

OptionDefListType SPackDefList;
OptionDefListType MBoxDefList;

void CScriptMDlg::LoadOptions( void )
{
	CString Line;
	OptionDefListType*	Cur = NULL;
	int l,p;
	char c;

	char fn[MAX_PATH];
	sprintf( fn, "sloptions.ini" );
	FILE* f = fopen( fn, "r" );
	if( f == NULL )
	{
//		MessageBox( "Cannot open Options.INI", "File Error", MB_OK |
MB_ICONWARNING );
		return;
	}

	fseek( f, 0, SEEK_END );
	l = ftell( f );
	fseek( f, 0, SEEK_SET );

	CString CurrentCat = "";

	while( ftell( f ) < l )
	{
		c = fgetc( f );
		Line = "";

		while( c != '\n' && ftell( f ) < l )
		{
			Line += c;
```

```
            c = fgetc( f );
        }

        if( Line == "[Message Box Defaults]" )
        {
            Cur = &MBoxDefList;
        }
        else
        if( Line == "[ServicePacks]" )
        {
            Cur = &SPackDefList;
        }
        else
        if( Line == "[Shortcuts]" )
        {
            Cur = &ShortcutDefList;
        }
        else
        if( Line.Left( 9 ) == "[MSOffice" )
        {
            Cur = &OfficeDefList;

            CurrentCat = Line.Mid( 10 );
            CurrentCat = CurrentCat.Left( CurrentCat.GetLength()-1 );

            bool used = false;

            for( int x=0;x<OfficeCats;x++ )
                if( OfficeCatList[ x ] == CurrentCat ) used = true;

            if( !used ) { OfficeCatList[ OfficeCats ] = CurrentCat;
OfficeCats++; };
        }
        else
        if( Line == "[ShellFolders]" )
        {
            Cur = &ShellDefList;
        }
        else
        if( Line == "[NTCommonShellFolders]" )
        {
            Cur = &NTCFDefList;
        }
        else
        if( Line.Left( 9 ) == "[Policies" )
        {
            Cur = &PolicyDefList;

            CurrentCat = Line.Mid( 10 );
            CurrentCat = CurrentCat.Left( CurrentCat.GetLength()-1 );

            bool used = false;
            for( int x=0;x<PolicyCats;x++ )
            {
```

```
                        if( PolicyCatList[x] == CurrentCat ) used = true;
                }

                if( !used ) { PolicyCatList[ PolicyCats ] = CurrentCat;
PolicyCats++; };
        }
        else
        if( Line.GetLength() == 0 )
        {
        }
        else
        if( Line[0] == ';' )
        {
        }
        else
        if( Cur == NULL )
        {
        }
        else
        {
                Cur->Def[Cur->Defs].Text = "";
                Cur->Def[Cur->Defs].OS = "";
                Cur->Def[Cur->Defs].Key = "";
                Cur->Def[Cur->Defs].Val = "";

                p = 0;
                Line[p];

                while( p < Line.GetLength() && Line[p] != ',' )
                {
                        Cur->Def[Cur->Defs].Text += Line[p];
                        p++;
                }
                p++;

                while( p < Line.GetLength() && Line[p] != ',')
                {
                        Cur->Def[Cur->Defs].OS += Line[p];
                        p++;
                }
                p++;

                // Add to != list if 3 entry item
                if( Cur != &NTCFDefList && Cur != &ShellDefList && Cur !=
&ShortcutDefList && Cur != &MBoxDefList )
                {
                        while( p < Line.GetLength() && Line[p] != ',' )
                        {
                                Cur->Def[Cur->Defs].Key += Line[p];
                                p++;
                        }
                        p++;
                }
```

```
            while( p < Line.GetLength() && Line[p] != ',' )
            {
                  Cur->Def[Cur->Defs].Val += Line[p];
                  p++;
            }

            if( Cur == &OfficeDefList )
            {
                  CString T = "";

                  Cur->Def[Cur->Defs].File = false;

                  p++;
                  while( p < Line.GetLength() && Line[p] != ',' )
                  {
                        T += Line[p];
                        p++;
                  }

                  T.TrimLeft();
                  T.TrimRight();
                  T.MakeUpper();

                  if( T == "FILE" )
                  {
                        Cur->Def[Cur->Defs].File = true;
                  }
            }

            Cur->Def[Cur->Defs].Category = CurrentCat;

            if( Cur->Defs < 254 )
            Cur->Defs++;
      }
}

fclose( f );

Cur = NULL;

sprintf( fn, "slcustopt.ini" );
f = fopen( fn, "r" );
if( f == NULL )
{
      return;
}

fseek( f, 0, SEEK_END );
l = ftell( f );
fseek( f, 0, SEEK_SET );

while( ftell( f ) < l )
{
      c = fgetc( f );
```

```
		Line = "";

		while( c != '\n' && ftell( f ) < l )
		{
			Line += c;
			c = fgetc( f );
		}

		if( Line == "[Message Box Defaults]" )
		{
			Cur = &MBoxDefList;
		}
		else
		if( Line == "[ServicePacks]" )
		{
			Cur = &SPackDefList;
		}
		else
		if( Line == "[Shortcuts]" )
		{
			Cur = &ShortcutDefList;
		}
		else
		if( Line.Left( 9 ) == "[MSOffice" )
		{
			Cur = &OfficeDefList;

			CurrentCat = Line.Mid( 10 );
			CurrentCat = CurrentCat.Left( CurrentCat.GetLength()-1 );

			bool used = false;

			for( int x=0;x<OfficeCats;x++ )
				if( OfficeCatList[ x ] == CurrentCat ) used = true;

			if( !used ) { OfficeCatList[ OfficeCats ] = CurrentCat;
OfficeCats++; };
		}
		else
		if( Line == "[ShellFolders]" )
		{
			Cur = &ShellDefList;
		}
		else
		if( Line == "[NTCommonShellFolders]" )
		{
			Cur = &NTCFDefList;
		}
		else
		if( Line.Left( 9 ) == "[Policies" )
		{
			Cur = &PolicyDefList;

			CurrentCat = Line.Mid( 10 );
```

```
                CurrentCat = CurrentCat.Left( CurrentCat.GetLength()-1 );

                bool used = false;
                for( int x=0;x<PolicyCats;x++ )
                {
                        if( PolicyCatList[x] == CurrentCat ) used = true;
                }

                if( !used ) { PolicyCatList[ PolicyCats ] = CurrentCat;
PolicyCats++; };
        }
        else
        if( Line.GetLength() == 0 )
        {
        }
        else
        if( Line[0] == ';' )
        {
        }
        else
        if( Cur == NULL )
        {
        }
        else
        {
                Cur->Def[Cur->Defs].Text = "";
                Cur->Def[Cur->Defs].OS = "";
                Cur->Def[Cur->Defs].Key = "";
                Cur->Def[Cur->Defs].Val = "";

                p = 0;
                Line[p];

                while( p < Line.GetLength() && Line[p] != ',' )
                {
                        Cur->Def[Cur->Defs].Text += Line[p];
                        p++;
                }
                p++;

                while( p < Line.GetLength() && Line[p] != ',')
                {
                        Cur->Def[Cur->Defs].OS += Line[p];
                        p++;
                }
                p++;

                // Add to != list if 3 entry item
                if( Cur != &NTCFDefList && Cur != &ShellDefList && Cur !=
&ShortcutDefList && Cur != &MBoxDefList )
                {
                        while( p < Line.GetLength() && Line[p] != ',' )
                        {
                                Cur->Def[Cur->Defs].Key += Line[p];
```

```
                        p++;
                }
                p++;
        }

        while( p < Line.GetLength() && Line[p] != ',' )
        {
                Cur->Def[Cur->Defs].Val += Line[p];
                p++;
        }

        if( Cur == &OfficeDefList )
        {
                CString T = "";

                Cur->Def[Cur->Defs].File = false;

                p++;
                while( p < Line.GetLength() && Line[p] != ',' )
                {
                        T += Line[p];
                        p++;
                }

                T.TrimLeft();
                T.TrimRight();
                T.MakeUpper();

                if( T == "FILE" )
                {
                        Cur->Def[Cur->Defs].File = true;
                }
        }

        Cur->Def[Cur->Defs].Category = CurrentCat;

        if( Cur->Defs < 254 )
        Cur->Defs++;
    }
}

fclose( f );

/*
while( ftell( f ) < l )
{
    c = fgetc( f );
    Line = "";

    while( c != '\n' && ftell( f ) < l )
    {
        Line += c;
        c = fgetc( f );
    }
```

```
if( Line == "[Shortcuts]" )
{
      Cur = &ShortcutDefList;
}
else
if( Line == "[ListLimits]" )
{
      Cur = NULL;
}
else
if( Line == "[MSOffice]" )
{
      Cur = &OfficeDefList;
}
else
if( Line == "[ShellFolders]" )
{
      Cur = &ShellDefList;
}
else
if( Line == "[NTCommonShellFolders]" )
{
      Cur = &NTCFDefList;
}
else
if( Line == "[Policies]" )
{
      Cur = &PolicyDefList;
}
else
if( Line.GetLength() == 0 )
{
}
else
if( Line[0] == ';' )
{
}
else
if( Cur == NULL )
{
}
else
{

      Cur->Def[Cur->Defs].Text = "";
      Cur->Def[Cur->Defs].OS = "";
      Cur->Def[Cur->Defs].Key = "";
      Cur->Def[Cur->Defs].Val = "";

      p = 0;
      Line[p];

      while( p < Line.GetLength() && Line[p] != ',' )
```

```
                    {
                          Cur->Def[Cur->Defs].Text += Line[p];
                          p++;
                    }
                    p++;

                    while( p < Line.GetLength() && Line[p] != ',')
                    {
                          Cur->Def[Cur->Defs].OS += Line[p];
                          p++;
                    }
                    p++;

                    if( Cur != &NTCFDefList && Cur != &ShellDefList && Cur !=
&ShortcutDefList )
                    {
                          while( p < Line.GetLength() && Line[p] != ',' )
                          {
                                Cur->Def[Cur->Defs].Key += Line[p];
                                p++;
                          }
                          p++;
                    }

                    while( p < Line.GetLength() && Line[p] != ',' )
                    {
                          Cur->Def[Cur->Defs].Val += Line[p];
                          p++;
                    }

                    if( Cur == &OfficeDefList )
                    {
                          CString T = "";

                          Cur->Def[Cur->Defs].File = false;

                          p++;
                          while( p < Line.GetLength() && Line[p] != ',' )
                          {
                                T += Line[p];
                                p++;
                          }

                          T.TrimLeft();
                          T.TrimRight();
                          T.MakeUpper();

                          if( T == "FILE" )
                          {
                                Cur->Def[Cur->Defs].File = true;
                          }
                    }

                    Cur->Def[Cur->Defs].Category = CurrentCat;
```

```
                    if( Cur->Defs < 254 )
                    Cur->Defs++;

/*
                    Cur->Def[Cur->Defs].Text = "";
                    Cur->Def[Cur->Defs].OS = "";
                    Cur->Def[Cur->Defs].Key = "";
                    Cur->Def[Cur->Defs].Val = "";

                    p = 0;
                    Line[p];

                    while( p < Line.GetLength() && Line[p] != ',')
                    {
                          Cur->Def[Cur->Defs].Text += Line[p];
                          p++;
                    }
                    p++;

                    while( p < Line.GetLength() && Line[p] != ',' )
                    {
                          Cur->Def[Cur->Defs].OS += Line[p];
                          p++;
                    }
                    p++;

                    if( Cur != &NTCFDefList && Cur != &ShellDefList )
                    {
                          while( p < Line.GetLength() && Line[p] != ',' )
                          {
                                Cur->Def[Cur->Defs].Key += Line[p];
                                p++;
                          }
                          p++;
                    }

                    while( p < Line.GetLength() && Line[p] != ',' )
                    {
                          Cur->Def[Cur->Defs].Val += Line[p];
                          p++;
                    }

                    Cur->Defs++;
               }
          }
*/


}

void CScriptMDlg::LoadMacroDefs( void )
{
      char  c;
```

```
	int		l;

	char fn[MAX_PATH];
	sprintf( fn, "slmacros.ini" );
	FILE* f = fopen( fn, "r" );
	if( f == NULL )
	{
//		MessageBox( "Cannot open macrodef.ini", "File Error", MB_OK |
MB_ICONWARNING );
		return;
	}

	fseek( f, 0, SEEK_END );
	l = ftell( f );
	fseek( f, 0, SEEK_SET );

	while( ftell( f ) < l )
	{
		c = fgetc( f );
		MacroDefList.MacroDef[MacroDefList.MacroDefs].OS = "";
		while( (c != '\n' && c != ',') && (ftell( f ) < l) )
		{
			MacroDefList.MacroDef[MacroDefList.MacroDefs].OS += c;
			c = fgetc( f );
		}

		c = fgetc( f );
		MacroDefList.MacroDef[MacroDefList.MacroDefs].Macro = "";
		while( (c != '\n' && c != ',') && (ftell( f ) < l) )
		{
			MacroDefList.MacroDef[MacroDefList.MacroDefs].Macro += c;
			c = fgetc( f );
		}

		c = fgetc( f );
		MacroDefList.MacroDef[MacroDefList.MacroDefs].Text = "";
		while( (c != '\n' && c != ',') && (ftell( f ) < l) )
		{
			MacroDefList.MacroDef[MacroDefList.MacroDefs].Text += c;
			c = fgetc( f );
		}


		MacroDefList.MacroDefs++;
	}

	fclose( f );
}

void DoubleAmps( CString& s )
{
	int x;

	for( x=0;x<s.GetLength();x++ )
```

```
        {
                if( s[x] == '&' )
                {
                        s = s.Left(x) + '&' + s.Right(s.GetLength()-x);
                        x++;
                }
        }
}

bool GetBitmapAndPalette(UINT nIDResource, CBitmap &bitmap, CPalette &pal)
{

        LPCTSTR lpszResourceName = MAKEINTRESOURCE(nIDResource);

        HBITMAP hBmp = (HBITMAP)::LoadImage( AfxGetResourceHandle(),
                        lpszResourceName, IMAGE_BITMAP, 0,0,
LR_CREATEDIBSECTION );

        if( hBmp == NULL )
                return FALSE;

        bitmap.Attach( hBmp );

        // Create a logical palette for the bitmap
        DIBSECTION ds;
        BITMAPINFOHEADER &bmInfo = ds.dsBmih;
        bitmap.GetObject( sizeof(ds), &ds );

        int nColors = bmInfo.biClrUsed ? bmInfo.biClrUsed : 1 <<
bmInfo.biBitCount;

        // Create a halftone palette if colors > 256.
        CClientDC dc(NULL);                     // Desktop DC
        if( nColors > 256 )
            {
                pal.CreateHalftonePalette( &dc );
            }
        else
        {
                // Create the palette

                RGBQUAD *pRGB = new RGBQUAD[nColors];
                CDC memDC;
                CBitmap memBitmap;
                CBitmap* oldBitmap;
                memDC.CreateCompatibleDC(&dc);
                memBitmap.CreateCompatibleBitmap(&dc,
bitmap.GetBitmapDimension().cx, bitmap.GetBitmapDimension().cy);

                oldBitmap = (CBitmap*)memDC.SelectObject( &memBitmap );
                ::GetDIBColorTable( memDC, 0, nColors, pRGB );

                UINT nSize = sizeof(LOGPALETTE) + (sizeof(PALETTEENTRY) *
nColors);
```

```
                LOGPALETTE *pLP = (LOGPALETTE *) new BYTE[nSize];

                pLP->palVersion = 0x300;
                pLP->palNumEntries = nColors;

                for( int i=0; i < nColors; i++)
                {
                        pLP->palPalEntry[i].peRed = pRGB[i].rgbRed;
                        pLP->palPalEntry[i].peGreen = pRGB[i].rgbGreen;
                        pLP->palPalEntry[i].peBlue = pRGB[i].rgbBlue;
                        pLP->palPalEntry[i].peFlags = 0;
                }

                pal.CreatePalette( pLP );

                delete[] pLP;
                delete[] pRGB;

                memDC.SelectObject(oldBitmap);
        }

        return TRUE;
}

extern void TrimString( CString& s );

void LoadDefaults( void )
{
      ParseINI pi( true );

      CString t;

      sprintf( DefaultGroup, "*" );
      Default95 = true;
      Default98 = true;
      DefaultNT = true;
      DefaultLAN = true;
      DefaultRAS = true;
      PromptDelete = true;

      char t1[256];

      sprintf( t1, "" );
      if( pi.OpenINI( "slmgr.ini" ) )
      {
            pi.SkipWhite = false;
            pi.FindValue( "LastTab", t1 );
            if( strcmp( t1," " ) )
            sprintf( LastPage, "%s", t1 );
            pi.SkipWhite = true;

            pi.FindValue( "DefaultGroup", t1 );
            if( strcmp( t1," " ) )
            sprintf( DefaultGroup, "%s", t1 );
```

```
pi.FindValue( "PromptDelete", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) PromptDelete = false;

pi.FindValue( "DefaultValType", t1 );
t = t1;
t.MakeUpper();
char* tcp;
DefaultVType = strtol( (LPCTSTR)t, &tcp, 10 );

pi.FindValue( "Default95", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) Default95 = false;

pi.FindValue( "Default98", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) Default98 = false;

pi.FindValue( "DefaultNT", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) DefaultNT = false;

pi.FindValue( "DefaultTSC", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) DefaultTSC = false;

pi.FindValue( "DefaultNTS", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) DefaultNTS = false;

pi.FindValue( "DefaultLAN", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) DefaultLAN = false;

pi.FindValue( "ListHidden", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) ListHidden = false;

pi.FindValue( "DefaultRAS", t1 );
t = t1;
t.MakeUpper();
if( t == "NO" ) DefaultRAS = false;

pi.FindValue( "UserGroup", t1 );
if( strcmp( t1, " " ) )
```

```
            SysLangUsers = t1;

            pi.FindValue( "EveryoneGroup", t1 );
            if( strcmp( t1, " " ) )
            SysLangEveryone = t1;

            pi.FindValue( "DomainAdminGroup", t1 );
            if( strcmp( t1, " " ) )
            SysLangDomainAdmins = t1;

            pi.FindValue( "AdministratorsGroup", t1 );
            if( strcmp( t1, " " ) )
            SysLangAdministrators = t1;

            pi.FindValue( "Enumeration", t1 );
            if( strcmp( t1, " " ) )
            {
                  if( strcmp( t1, "0" ) == 0 )
                  {
                        Enumeration = false;
                  }
            }

            SysLangUsers.TrimLeft();
            SysLangUsers.TrimRight();
            SysLangEveryone.TrimLeft();
            SysLangEveryone.TrimRight();
            SysLangDomainAdmins.TrimLeft();
            SysLangDomainAdmins.TrimRight();
            SysLangAdministrators.TrimLeft();
            SysLangAdministrators.TrimRight();
      }
}

void SaveDefaults( void )
{
      INIFileWrapper slini( "slmgr.ini", true );

      slini.SetValue( "DefaultGroup", DefaultGroup, "Defaults" );

      char temps[256];
      char* tcp;
      _itoa( DefaultVType, temps, 10 );

      slini.SetValue( "DefaultValType", temps, "Defaults" );

      if( PromptDelete )
      {
            slini.SetValue( "PromptDelete", "Yes", "Defaults" );
      }
      else
      {
            slini.SetValue( "PromptDelete", "No", "Defaults" );
      }
```

```
if( Default95 )
{
      slini.SetValue( "Default95", "Yes", "Defaults" );
}
else
{
      slini.SetValue( "Default95", "No", "Defaults" );
}

if( Default98 )
{
      slini.SetValue( "Default98", "Yes", "Defaults" );
}
else
{
      slini.SetValue( "Default98", "No", "Defaults" );
}

if( DefaultNT )
{
      slini.SetValue( "DefaultNT", "Yes", "Defaults" );
}
else
{
      slini.SetValue( "DefaultNT", "No", "Defaults" );
}

if( DefaultNTS )
{
      slini.SetValue( "DefaultNTS", "Yes", "Defaults" );
}
else
{
      slini.SetValue( "DefaultNTS", "No", "Defaults" );
}

if( DefaultTSC )
{
      slini.SetValue( "DefaultTSC", "Yes", "Defaults" );
}
else
{
      slini.SetValue( "DefaultTSC", "No", "Defaults" );
}

if( ListHidden )
{
      slini.SetValue( "ListHidden", "Yes", "Defaults" );
}
else
{
      slini.SetValue( "ListHidden", "No", "Defaults" );
}
```

```
	if( DefaultLAN )
	{
		slini.SetValue( "DefaultLAN", "Yes", "Defaults" );
	}
	else
	{
		slini.SetValue( "DefaultLAN", "No", "Defaults" );
	}

	if( DefaultRAS )
	{
		slini.SetValue( "DefaultRAS", "Yes", "Defaults" );
	}
	else
	{
		slini.SetValue( "DefaultRAS", "No", "Defaults" );
	}
}

void SaveTargetList( void )
{
	INIFileWrapper slini( "slmgr.ini", true );

	int TN = Targets;

	for( int x=0;x<256;x++ )
	{
		char temps[256];
		sprintf( temps, "Target%i", x );
		slini.DeleteValue( temps );
	}

	for( x = 0; x<Targets; x++ )
	{
		char temps[256];
		sprintf( temps, "Target%i", TN );

		TN--;

		slini.SetValue( temps, TargetList[x], "Targets" );
	}

	slini.SetValue( "Scripts", DefaultTarget, "File Locations" );
}

void LoadTargetList( void )
{
	ParseINI pi( true );

	CString t;

	Targets = 0;
```

```
	char t1[256];

	sprintf( t1, "" );
	INIToken	tok;
	if( pi.OpenINI( "slmgr.ini" ) )
	{
		while( !pi.atEnd )
		{
			tok = pi.NextToken();

			if( tok.Type == INI_TOKEN_VARIABLE )
			{
				CString l;

				l = tok.Name;
				l = l.Left( 6 );
				l.MakeUpper();

				if( l == "TARGET" )
				{
					TargetList[Targets] = tok.Value;
					Targets++;
				}
			}
		}
	}
}

void GetASmgrPath()
{
	HKEY k;

	DWORD disp;

	if( RegCreateKeyEx( HKEY_LOCAL_MACHINE, "Software\\Inteletek\\AutoShare",
0, NULL, REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &k, &disp ) ==
ERROR_SUCCESS )
	{
		if( disp == REG_OPENED_EXISTING_KEY )
		{
			char path[MAX_PATH];
			DWORD size = MAX_PATH;

			if( RegQueryValueEx( k, "MangerPath", NULL, NULL, (unsigned
char*)(path), &size ) == ERROR_SUCCESS )
			{
				ASmgrPath = path;
			}
		}

		RegCloseKey( k );
	}
}
```

```
CString UserLic;
extern DWORD dwNumLic;
CString LogonServer;

void GetUserInfo()
{
	static DWORD count;
	static char buf[256];
	static char* b;
	count = 256;

	HKEY DisplayKey;

	RegOpenKeyEx( HKEY_CURRENT_USER, "Software\\ScriptLogic", 0,
KEY_ALL_ACCESS, &DisplayKey );

	if( RegQueryValueEx( DisplayKey, "UserLicense", 0, NULL, (LPBYTE)buf,
&count ) == ERROR_SUCCESS )
	{
		// TODO : if license's <= 0 then Evaluation Edition
		UserLic = buf;
	}
	else
	{
		UserLic = "Evaluation Edition";
	}

	count = 256;
	if( RegQueryValueEx( DisplayKey, "LogonServer", 0, NULL, (LPBYTE)buf,
&count ) == ERROR_SUCCESS )
	{
		LogonServer = buf;
	}
	else
	{
		LogonServer = "";
	}

	count = sizeof( DWORD );

	if( RegQueryValueEx( DisplayKey, "Licenses", 0, NULL,
reinterpret_cast<unsigned char*>(&dwNumLic), &count ) == ERROR_SUCCESS )
	{
	}
	else
	{
		dwNumLic = -1;
	}

	RegCloseKey( DisplayKey );
}

BOOL CScriptMDlg::OnInitDialog()
{
```

```
    b1.AutoLoad( IDC_BITMAP2, this );

  GetBitmapAndPalette( IDB_BITMAP17, MyBitmap, MyPalette );

    if( nosave )
    {
          m_Bottom = "Log-View Only";
    }
    else
    {
          m_Bottom = "Script Folder: ";
          m_Bottom += ScriptPath;
    }

    m_Top = "Version 3.03 Pro";

    oxDib.LoadResource( MAKEINTRESOURCE(IDB_BITMAP17), FALSE );

    CDialog::OnInitDialog();

//  m_SS.LoadBitmap( MAKEINTRESOURCE( IDB_BITMAP17 ) );

    m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_GREEN ), FALSE );

    //EnumerateFunc( NULL );

    NotepadHandle = NULL;
    NotepadHandle2 = NULL;
    SMHandle = NULL;

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
          CString strAboutMenu;
          strAboutMenu.LoadString(IDS_ABOUTBOX);
          if (!strAboutMenu.IsEmpty())
          {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
          }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);             // Set big icon
    SetIcon(m_hIcon, FALSE);            // Set small icon

    SS2.PrintMessage( "Loading Defaults..." );
```

```
LoadDefaults();

SS2.PrintMessage( "Loading Options..." );

LoadOptions();

SS2.PrintMessage( "Loading Macro Definitions..." );

LoadMacroDefs();

SS2.PrintMessage( "Loading Target List..." );

LoadTargetList();

GetUserInfo();

GetASmgrPath();

bool SPfound = false;

for( int x=0;x<Targets;x++ )
{
      CString t1,t2;
      t1 = TargetList[x];
      t2 = ScriptPath;

      t1.MakeUpper();
      t2.MakeUpper();

      if( t1 == t2 ) SPfound = true;
      m_LocationC.AddString( TargetList[x] );
}

if( !SPfound )
{
      TargetList[x] = ScriptPath;
      Targets++;
      m_LocationC.AddString( TargetList[x] );
}

m_Location = ScriptPath;

SS2.PrintMessage( "Loading Configuration..." );

S.LoadScript( );
S.SaveScript( "SLconfig.bak" );

char t1[256];
char t2[256];
sprintf( t1, "%s\\slconfig.bak", (LPCTSTR)ScriptPath );
sprintf( t2, "%s\\sltest.val", (LPCTSTR)ScriptPath );
CopyFile( t1, t2, FALSE );
```

```
	if( ASmgrPath != "" ) m_ASmgr.EnableWindow( TRUE );

	m_SS.EnableWindow( FALSE );


//	m_User.SubclassDlgItem( IDC_KEY2, this );
//	m_User.SetTextColor( RGB( 0,0,0 ) );
//	m_User.SetBkColor( RGB( 255,255,255 ) );

//	m_Company.SubclassDlgItem( IDC_COMPANY, this );
//	m_Company.SetTextColor( RGB( 0,0,0 ) );
//	m_Company.SetBkColor( RGB( 255,255,255 ) );
//	m_Company.SetBkColor( RGB( 20,0,130 ) );

//	m_Key.SubclassDlgItem( IDC_KEY, this );
//	m_Key.SetTextColor( RGB( 0,0,0 ) );
//	m_Key.SetBkColor( RGB( 255,255,255 ) );
//	m_Key.SetBkColor( RGB( 20,0,130 ) );


	DoubleAmps( CompanyName );
	DoubleAmps( RegKey );
//	((CStatic*)GetDlgItem( IDC_COMPANY ))->SetWindowText( CompanyName );
//	((CStatic*)GetDlgItem( IDC_KEY ))->SetWindowText( RegKey );
//	((CStatic*)GetDlgItem( IDC_KEY2 ))->SetWindowText( UserLic );

	UpdateData( FALSE );

	EnumerateLocalGroups();

	SetTimer( 1, 500, NULL );

	char temps[256];
	sprintf( temps, "%s\\slconfig.lck", ScriptPath );

	char cname[256];
	char uname[256];
	sprintf( cname, "" );
	sprintf( uname, "" );
	GetEnvironmentVariable( "UserName", uname, 256 );
	GetEnvironmentVariable( "ComputerName", cname, 256 );

	SS2.PrintMessage( "Updating Lock File..." );

	sprintf( temps, "%s\\slconfig.lck", ScriptPath );
	if( GetFileAttributes( temps ) != 0xFFFFFFFF )
	{
//		ScriptFileHandle = CreateFile( temps, GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );
		FILE* f = fopen( temps, "r" );
		CString ts2 = "another administrator on the network";

		if( f == NULL )
		{
```

```
            }
            else
            {
                  fseek( f, 0, SEEK_END );
                  int l = ftell( f );
                  fseek( f, 0, SEEK_SET );
                  ts2 = "";
                  for( int x=0;x<l;x++ )
                  {
                        ts2 += fgetc( f );
                  }
                  fclose( f );
            }


            DWORD br;
//          ReadFile( ScriptFileHandle, temps, GetFileSize( ScriptFileHandle,
NULL ), &br, NULL );
            CString ts = ResString( IDS_EDITEDBY1 );
            ts += ts2;
            ts += ResString( IDS_EDITEDBY2 );
            SS2.Show(1, NULL );
//          SS.Stop();
            MessageBox( (LPCTSTR)ts, ResString( IDS_WARNING ), MB_OK |
MB_ICONHAND );
      }
      else
      {
            ScriptFileHandle = CreateFile( temps, GENERIC_WRITE | GENERIC_READ,
FILE_SHARE_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_HIDDEN, NULL );
            sprintf( temps, "%s on %s", uname, cname );
            DWORD bw;
            WriteFile( ScriptFileHandle, temps, strlen( temps ), &bw, NULL );
            CloseHandle( ScriptFileHandle );
      }

      if( ScriptPath.Right( strlen( "import\\scripts" ) ).CompareNoCase(
"import\\scripts" ) == 0 )
      {
            runbenabled = false;
            m_runrb.EnableWindow( FALSE );
//          m_editrb.EnableWindow( FALSE );
      }

      if( GetFileAttributes( (LPCTSTR)(ScriptPath + "\\repl.bat") ) ==
0xFFFFFFFF )
      {
            runbenabled = false;
            m_runrb.EnableWindow( FALSE );
      }

      if( nosave )
      {
            m_SUB.EnableWindow( FALSE );
```

```
            m_SS.EnableWindow( FALSE );
            m_editrb.EnableWindow( FALSE );
            runbenabled = false;
            m_runrb.EnableWindow( FALSE );
            m_cs.EnableWindow( FALSE );
            m_b1.EnableWindow( FALSE );
      }

      SS2.Show(1, NULL );
//    SS.Stop();

      if( ForceDirty )
      {
            ExitState = EXIT_NOSAVE;
            m_EB.RemoveImage( FALSE );
            m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_RED ), FALSE );
            m_EB.Invalidate();
            m_SS.EnableWindow( TRUE );
      }


      return TRUE;  // return TRUE  unless you set the focus to a control
}

void CScriptMDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
      if ((nID & 0xFFF0) == IDM_ABOUTBOX)
      {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
      }
      else
      {
            CDialog::OnSysCommand(nID, lParam);
      }
}


int FileSize( FILE* f )
{
      fseek( f, 0, SEEK_END );
      int l = ftell( f );
      fseek( f, 0, SEEK_SET );
      return l;
}

int FileSame( char* fn1, char* fn2 )
{
      FILE* f1 = fopen( fn1, "rb" );
      FILE* f2 = fopen( fn2, "rb" );

      if( f1 == NULL ) return true;
      if( f2 == NULL ) return true;
```

```
	int l1,l2;

	l1 = FileSize( f1 );
	l2 = FileSize( f2 );

	if( l1 != l2 )
	{
		fclose( f1 );
		fclose( f2 );
		return false;
	}

	int x;

	for( x=0;x<l1;x++ )
	{
		if( fgetc( f1 ) != fgetc( f2 ) )
		{
			fclose( f1 );
			fclose( f2 );
			return false;
		}
	}


	fclose( f1 );
	fclose( f2 );
	return true;
}

bool ForceDirty = false;

int CScriptMDlg::isDirty( void )
{
	if( nosave ) return false;

	if( ForceDirty ) return true;

	char t1[256];
	char t2[256];

	sprintf( t1, "%s\\temp.out", (LPCTSTR)ScriptPath );
	sprintf( t2, "%s\\sltest.val", (LPCTSTR)ScriptPath );

	S.SaveScript( "temp.out" );

	if( GetFileAttributes( t2 ) == 0xFFFFFFFF ) return true;

	if( FileSame( t1, t2 ) )
	{
		DeleteFile( t1 );
		return false;
	}
```

```
	DeleteFile( t1 );

	ExitState = EXIT_NOSAVE;
	m_EB.RemoveImage( FALSE );
	m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_RED ), FALSE );
	m_EB.Invalidate();

	return true;
}


// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.
bool PlaceBitmap(CDC* dc, UINT nIDResource, int x, int y,int xs, int ys)
{
	// Create a memory DC compatible with the paint DC
	CDC memDC;
	CBitmap memBitmap;
	CBitmap* oldBitmap;
	memDC.CreateCompatibleDC(dc);
	memBitmap.CreateCompatibleBitmap(dc, xs, ys);

	CBitmap bitmap;
	CPalette palette;

	GetBitmapAndPalette( nIDResource, bitmap, palette );
	oldBitmap = (CBitmap*)memDC.SelectObject(&bitmap);

	// Select and realize the palette
	if( dc->GetDeviceCaps(RASTERCAPS) & RC_PALETTE && palette.m_hObject !=
NULL )
	{
		dc->SelectPalette( &palette, FALSE );
		dc->RealizePalette();
	}

	dc->BitBlt(x, y, xs, ys, &memDC, 0, 0,SRCCOPY);

	memDC.SelectObject(oldBitmap);
	return TRUE;
}

void CScriptMDlg::OnPaint()
{

	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
```

```
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;

            // Draw the icon
            dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
            CPaintDC dc(this);

            RECT r;
            r.top = 6;
            r.left = 14;
            r.bottom = 40;
            r.right = 40;
            MapDialogRect( &r );

            int x = r.left;
            int y = r.top;
            int xs = 202;
            int ys = 214;

/*
          CDC memDC;
          CBitmap memBitmap;
          CBitmap* oldBitmap;
          memDC.CreateCompatibleDC(&dc);
          memBitmap.CreateCompatibleBitmap(&dc, xs, ys);

          oldBitmap = (CBitmap*)memDC.SelectObject(&MyBitmap);

          // Select and realize the palette
          if( dc.GetDeviceCaps(RASTERCAPS) & RC_PALETTE && MyPalette.m_hObject !=
NULL )
          {
                  dc.SelectPalette( &MyPalette, TRUE );
                  dc.RealizePalette();
          }

          dc.BitBlt(x, y, xs, ys, &memDC, 0, 0,SRCCOPY);

          memDC.SelectObject(oldBitmap);
*/
            oxDib.Draw( &dc, x,y );

            CFont*      OF;
            CFont CF;
            CF.CreatePointFont( 80, "MS Sans Serif" );

            OF = dc.SelectObject( &CF );
```

```
            // 37,78
            // 37,86
            // 37,94

//    ((CStatic*)GetDlgItem( IDC_COMPANY ))->SetWindowText( CompanyName );
//    ((CStatic*)GetDlgItem( IDC_KEY ))->SetWindowText( RegKey );
//    ((CStatic*)GetDlgItem( IDC_KEY2 ))->SetWindowText( UserLic );

            dc.SetTextColor( RGB(0,0,0) );
            dc.SetBkMode( TRANSPARENT );

            r.top = 83;
            r.left = 21;
            r.bottom = 140;
            r.right = 142;
            MapDialogRect( &r );
            x = r.left;
            y = r.top;
            dc.ExtTextOut( x,y, ETO_CLIPPED, &r, CompanyName,
CompanyName.GetLength(), NULL );

            r.top = 93;
            r.left = 21;
            r.bottom = 140;
            r.right = 142;
            MapDialogRect( &r );
            x = r.left;
            y = r.top;
            dc.ExtTextOut( x,y, ETO_CLIPPED, &r, UserLic, UserLic.GetLength(),
NULL );
            /*
            r.top = 92;
            r.left = 37;
            r.bottom = 140;
            r.right = 140;
            MapDialogRect( &r );
            x = r.left;
            y = r.top;
            dc.TextOut( x,y, UserLic );
*/
            dc.SelectObject( OF );

            CDialog::OnPaint();

      }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CScriptMDlg::OnQueryDragIcon()
{
      return (HCURSOR) m_hIcon;
}
```

```
int CScriptMDlg::DoModal()
{
	return CDialog::DoModal();
}

void CScriptMDlg::OnButton1()
{
	S.UpdateSheet( PS );
	PS.DoModal();
	S.UpdateScript( PS );

	if( isDirty( ) )
	{
		ExitState = EXIT_NOSAVE;
		m_EB.RemoveImage( FALSE );
		m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_RED ), FALSE );
		m_EB.Invalidate();
		m_SS.EnableWindow( TRUE );
	}
}

void CScriptMDlg::OnButton2()
{
	// TODO: Add your control notification handler code here

	int r;
	char t1[256];

	int dir = true;

	if( isDirty() )
	if( (r = MessageBox( ResString( IDS_SAVE ), ResString( IDS_SLMGR ),
MB_YESNOCANCEL | MB_ICONWARNING )) == IDYES )
	{
		dir = true;
		Repl = false;
		ForceDirty = false;
		S.SaveScript();
		m_SS.EnableWindow( FALSE );
		S.SaveScript( "sltest.val" );

		if( OnSaveCommand != "" )
		{
			system( (LPCTSTR)OnSaveCommand );
		}
	}
	else
	{
		dir = false;
	}

	if( r == IDCANCEL ) return;

	CString SP = ScriptPath;
```

```
      SP.MakeUpper();
//    if( !Repl && (SP.Right( strlen( "Export\\Scripts" ) ) ==
"EXPORT\\SCRIPTS" ))

//    if( dir && !Repl )
      if( ExitState == EXIT_NOSAVE || ExitState == EXIT_NOPUB )
      if( GetFileAttributes( (LPCTSTR)(ScriptPath + "\\repl.bat") ) !=
0xFFFFFFFF )
      if( ScriptPath.Right( strlen( "import\\scripts" ) ).CompareNoCase(
"import\\scripts" ) != 0 )
      {
            if( (r = MessageBox( ResString( IDS_REPL ), ResString( IDS_SLMGR ),
MB_YESNOCANCEL | MB_ICONWARNING )) == IDYES )
            {
                  OnReplbatch();
            }
      }

      if( r == IDCANCEL )
      {
            return;
      }

      CloseHandle( ScriptFileHandle );

      sprintf( t1, "%s\\sltest.val", (LPCTSTR)ScriptPath );
      DeleteFile( t1 );

      sprintf( t1, "%s\\slconfig.lck", (LPCTSTR)ScriptPath );
      DeleteFile( t1 );

      INIFileWrapper slini( "slmgr.ini", true );
      CString lt = LastPage;
      slini.SetValue( "LastTab", lt, "Defaults" );
      slini.Close();

      OnCancel();
}

extern CString OnSaveCommand;

void CScriptMDlg::OnSavescript()
{
      char temps2[256];
      sprintf( temps2, "%s\\slogic.bat", (LPCTSTR)ScriptPath );

      CFile* nf;
      CFileStatus cfs;
      CTime CT = CTime::GetCurrentTime();

      if( (GetFileAttributes( temps2 ) != 0xFFFFFFFF) && ((GetFileAttributes(
temps2 ) & FILE_ATTRIBUTE_DIRECTORY) == 0) )
      {
            nf = NULL;
```

```
            nf = new CFile( temps2, CFile::modeReadWrite );

            if( nf != NULL )
            {
                  nf->GetStatus( cfs );
                  cfs.m_mtime = CT;
                  delete nf;

                  CFile::SetStatus( temps2, cfs );
            }
      }

      ForceDirty = false;

      if( ExitState == EXIT_NOSAVE )
      {
            ExitState = EXIT_NOPUB;
            m_EB.RemoveImage( FALSE );
            m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_YELLOW ), FALSE );
            m_EB.Invalidate();
      }

      Repl = false;

      ForceDirty = false;
      S.SaveScript();
      m_SS.EnableWindow( FALSE );
      S.SaveScript( "sltest.val" );
      if( OnSaveCommand != "" )
      {
                  system( (LPCTSTR)OnSaveCommand );
      }
      dirty = false;
}

void CScriptMDlg::OnClose()
{
      int r = IDOK;

      int dir = true;

      if( isDirty() )
      if( (r = MessageBox( ResString( IDS_SAVE ), ResString( IDS_SLMGR ),
MB_YESNOCANCEL | MB_ICONWARNING )) == IDYES )
      {
            if( ExitState == EXIT_NOSAVE )
            {
                  ExitState = EXIT_NOPUB;
                  m_EB.RemoveImage( FALSE );
                  m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_YELLOW ), FALSE );
                  m_EB.Invalidate();
            }

            dir = true;
```

```
            Repl = false;
            ForceDirty = false;
            S.SaveScript();
            m_SS.EnableWindow( FALSE );
            S.SaveScript( "sltest.val" );
            if( OnSaveCommand != "" )
            {
                  system( (LPCTSTR)OnSaveCommand );
            }
      }
      else
      {
            dir = false;
      }

      if( r == IDCANCEL ) return;

      CString SP = ScriptPath;
      SP.MakeUpper();
//    if( !Repl && (SP.Right( strlen( "Export\\Scripts" ) ) ==
"EXPORT\\SCRIPTS" ))

//    if( dir && !Repl )
      if( ExitState == EXIT_NOSAVE || ExitState == EXIT_NOPUB )
      if( GetFileAttributes( (LPCTSTR)(ScriptPath + "\\repl.bat") ) !=
0xFFFFFFFF )
      if( ScriptPath.Right( strlen( "import\\scripts" ) ).CompareNoCase(
"import\\scripts" ) != 0 )
      {
            if( (r = MessageBox( ResString( IDS_REPL ), ResString( IDS_SLMGR ),
MB_YESNOCANCEL | MB_ICONWARNING )) == IDYES )
            {
                  OnReplbatch();
            }
      }

      if( r == IDCANCEL )
      {
            return;
      }

      CloseHandle( ScriptFileHandle );

      char t1[256];
      sprintf( t1, "%s\\sltest.val", (LPCTSTR)ScriptPath );
      DeleteFile( t1 );

      sprintf( t1, "%s\\slconfig.lck", (LPCTSTR)ScriptPath );
      DeleteFile( t1 );

      INIFileWrapper slini( "slmgr.ini", true );
      CString lt = LastPage;
      slini.SetValue( "LastTab", lt, "Defaults" );
      slini.Close();
```

```
	CDialog::OnClose();
	PostMessage( WM_QUIT );
}

#include "PickPath.h"
#include "PW3.h"

extern Script S;

int	GetMonths( CString& fn )
{
	int p = fn.ReverseFind( '\\' );
	if( p == -1 ) p = 0;
	CString year = fn.Mid(p+1,4);
	CString month = fn.Mid(p+5,2);

	char* tcp;

//	MessageBox( NULL,fn, year, MB_OK );
	int ret = 12*(strtol((LPCTSTR)year,&tcp,10));
	ret += (strtol((LPCTSTR)month,&tcp,10));

	return ret;
}

void CScriptMDlg::OnReviewlog()
{

	// New ActiveX Calendar Chooser

	LogView* LV = NULL;
	MList* ML = NULL;
	PickD2* PD = NULL;
	PickPath* PP = NULL;
	char temps[256];
	int x;

	S.RetrievePwoDrive( "LogShare", LogPath );

	if( LogPath.GetLength() > 2 )
	{
		if( LogPath[LogPath.GetLength()-1] == '$' &&
LogPath[LogPath.GetLength()-2] == '$' )
		{
			LogPath = LogPath.Left( LogPath.GetLength()-1 );
		}
	}

	if( GetFileAttributes( (LPCTSTR)LogPath ) == 0xFFFFFFFF )
	{
		PP = new PickPath();

		PP->VerifyPath = true;
```

```
            PP->CheckForFile = false;
            PP->VerifyReadAccess = false;

            if( nosave )
            {
                  sprintf( temps, ResString( IDS_FIND_LOGS ) );
            }
            else
            {
                  sprintf( temps, ResString( IDS_INVALID_LOG ),
(LPCTSTR)LogPath );
            }

            PP->m_Message = temps;

            if( PP->DoModal() != IDCANCEL )
            {
                  LogPath = PP->m_Path;

                  if( !nosave )
                  {
                        if( MessageBox( ResString( IDS_UPDATE_LOG ), ResString(
IDS_UPDATE_SCRIPT ), MB_YESNO ) == IDYES )
                        {
                              S.SetPwoDrive( "LogShare", LogPath,
HEADING_LOGGING );
                        }
                  }
            }
            else
            {
                  return;
            }

            delete PP;
      }

      CString LP;

      LP = LogPath;

      if( LP[LP.GetLength()-1] != '\\' ) LP += '\\';

      CString Pre = LP;

      LP += "*.*";

      WIN32_FIND_DATA wfd;
      HANDLE fd;
      fd = FindFirstFile( (LPCTSTR)LP, &wfd );

      bool Purge = true;
      bool Asked = false;
```

```
	PW3*	PWD=NULL;


	if( fd != INVALID_HANDLE_VALUE )
	{
		bool done = 1;
		CFile* nf;
		CFileStatus cfs;
		CTime CT = CTime::GetCurrentTime();
		int logage;

		while( done && Purge )
		{
			if( (GetFileAttributes( Pre+wfd.cFileName ) != 0xFFFFFFFF) &&
((GetFileAttributes( Pre+wfd.cFileName ) & FILE_ATTRIBUTE_DIRECTORY) == 0) )
			{
				nf = new CFile( Pre+wfd.cFileName, CFile::modeRead );

				CString fn = Pre+wfd.cFileName;
				fn.MakeUpper();

//				if( fn.GetLength() == 12 )
				if( fn.Right(4) == ".CSV" )
				{
//					CTimeSpan ts;

//					nf->GetStatus( cfs );
//					ts = CT - cfs.m_ctime;

					S.RetrieveNumber( "RetainLogFileMonths", logage
);

//					char tes[1024];
//					sprintf( tes, "%s %i %i", (LPCTSTR)fn, GetMonths(
fn ), (CT.GetMonth()+(CT.GetYear()*12)) );
//					MessageBox( tes, "HI", MB_OK );

					if( (((CT.GetMonth()+(CT.GetYear()*12))-
GetMonths( fn )) > logage) && (logage > 0) )
					{
						if( Purge && !Asked)
						{
							Asked = true;

							if( MessageBox( ResString(
IDS_OLD_LOGS ), ResString( IDS_PURGE_LOGS ), MB_YESNO ) == IDNO )
							{
								Purge = false;
							}
							else
							{
								PWD = new PW3;
								PWD->Create( IDD_PLEASEWAIT3 );
							}
```

```
                                        }

                                        nf->Close();

                                        if( Purge )
                                        {
                                              DeleteFile( Pre+wfd.cFileName );
//                                            if( MessageBox( Pre+wfd.cFileName,
"Del", MB_OKCANCEL ) == IDCANCEL ) return;
                                        }
                                  }
                            }

                            if( nf )
                            delete nf;
                      }

                      done = FindNextFile( fd, &wfd );
                }
          }

          if( PWD != NULL ) PWD->DestroyWindow();

          PD = new PickD2();

          while( PD->DoModal() != IDCANCEL )
          {
                SetCursor( LoadCursor( NULL, IDC_WAIT ) );

                LV = new LogView();

                for( x=0;x<PD->fnames;x++ )
                {
                      sprintf( LV->fnlist[x], "%s", PD->fnlist[x] );
                }

                LV->fnames = PD->fnames;

                char temps[256];

                LV->DoModal();

                SetCursor( LoadCursor( NULL, IDC_ARROW ) );

                delete LV;
          };

          delete PD;

          return;

          /* Old Style Month List

          ML = new MList();
```

```
	if( ML == NULL ) MessageBox( "N", "N", MB_OK );

	if( ML->DoModal() == -1 ) MessageBox( "X", "X", MB_OK );

	if( ML->F[0] == 0 )
	{
		delete ML;
		return;
	}

	LV = new LogView();

	sprintf( LV->OpenFile, "%s", ML->F );

	LV->DoModal();

	delete ML;
	delete LV;

	*/
}

//void CScriptMDlg::OnNtvars()
//{
	//S.UpdateSheet2( PS2 );
	//PS2.DoModal();
	//S.UpdateScript2( PS2 );

	//if( isDirty( ) ) m_SS.EnableWindow( TRUE );
//}

#include "EditCustomScripts.h"

void CScriptMDlg::OnCustomscript()
{
	EditCustomScripts ecs;

	ecs.DoModal();

	if( isDirty( ) )
	{
		ExitState = EXIT_NOSAVE;
		m_EB.RemoveImage( FALSE );
		m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_RED ), FALSE );
		m_EB.Invalidate();
		m_SS.EnableWindow( TRUE );
	}

	/*
	DWORD ret;

	if( NotepadHandle == NULL )
	{
```

```
            ret = STILL_ACTIVE+1;
      }
      else
      {
            GetExitCodeProcess( NotepadHandle, &ret );
      }


      if( ret != STILL_ACTIVE )
      {
            char temps[256];
            char spath[MAX_PATH];
            if( !GetShortPathName( (LPCTSTR)ScriptPath, spath, MAX_PATH ) )
            {
                  sprintf( spath, "%s", (LPCTSTR)ScriptPath );
            }
            sprintf( temps, "%s\\SLCustom1.KIX", spath );

            char spath2[MAX_PATH];
            if( !GetShortPathName( (LPCTSTR)TextED, spath2, MAX_PATH ) )
            {
                  sprintf( spath2, "%s", (LPCTSTR)TextED );
            }

            NotepadHandle = (HANDLE)_spawnlp( _P_NOWAIT, spath2, spath2, temps,
NULL );
            dirty = true;

            if( ExitState == EXIT_OK )
            {
                  ExitState = EXIT_NOPUB;
                  m_EB.RemoveImage( FALSE );
                  m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_YELLOW ), FALSE );
                  m_EB.Invalidate();
            }
      }
*/
  }

void CScriptMDlg::OnStartupbat()
{
      OnToolsSystemoptions();

      /*
      DWORD ret;

      if( NotepadHandle2 == NULL )
      {
            ret = STILL_ACTIVE+1;
      }
      else
      {
            GetExitCodeProcess( NotepadHandle2, &ret );
      }
```

```
	if( ret != STILL_ACTIVE )
	{
		char temps[256];
		char spath[MAX_PATH];
		if( !GetShortPathName( (LPCTSTR)ScriptPath, spath, MAX_PATH ) )
		{
			sprintf( spath, "%s", (LPCTSTR)ScriptPath );
		}
		sprintf( temps, "%s\\SLCustom2.KIX", spath );

		char spath2[MAX_PATH];
		if( !GetShortPathName( (LPCTSTR)TextED, spath2, MAX_PATH ) )
		{
			sprintf( spath2, "%s", (LPCTSTR)TextED );
		}

		NotepadHandle2 = (HANDLE)_spawnlp( _P_NOWAIT, spath2, spath2,
temps, NULL );
		dirty = true;

		if( ExitState == EXIT_OK )
		{
			ExitState = EXIT_NOPUB;
			m_EB.RemoveImage( FALSE );
			m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_YELLOW ), FALSE );
			m_EB.Invalidate();
		}
	}
	*/
}

CString NextLine( FILE*, int );

CString	ReplBATList[256];
int		NumBATs;

void LoadBatch( void )
{
	FILE* rbat = fopen( ScriptPath + "\\repl.bat", "r" );

//	m_SourceFolder = "admin$\\system32\\repl\\export\\scripts";
//	m_DestFolder = "admin$\\system32\\repl\\import\\scripts";
//	m_Flags = "/h /r /d /c /i";
//	m_Pause = false;

	NumBATs = 0;

	if( rbat == NULL ) return;

	int l;

	fseek( rbat, 0, SEEK_END );
```

```
	l = ftell( rbat );
	fseek( rbat, 0, SEEK_SET );

	CString line;

	while( ftell( rbat ) < l )
	{
		line = NextLine( rbat, l );

//		if( line.Left(strlen( "pause" )) == "pause" ) m_Pause = true;

		if( line.Left(strlen( "set SourcePDC=")) == "set SourcePDC=" )
		{
			ReplBATList[NumBATs] = line.Right( line.GetLength()-
line.ReverseFind( '=' )-1 );
			if( NumBATs < 255 ) NumBATs++;
//			m_SourcePath = line.Right( line.GetLength()-strlen("set
SourcePDC=") );
		}

		if( line.Left(strlen( "set sourcefolder=")) == "set sourcefolder="
)
		{
//			m_SourceFolder = line.Right( line.GetLength()-strlen("set
sourcefolder=") );
		}

		if( line.Left(strlen( "set destfolder=")) == "set destfolder=" )
		{
//			m_DestFolder = line.Right( line.GetLength()-strlen("set
destfolder=") );
		}

		if( line.Left(strlen( "set copyflags=")) == "set copyflags=" )
		{
//			m_Flags = line.Right( line.GetLength()-strlen("set
copyflags=") );
		}

		if( line.Left(strlen( "set DestBDC")) == "set DestBDC" )
		{
			ReplBATList[NumBATs] = line.Right( line.GetLength()-
line.ReverseFind( '=' )-1 );
			if( NumBATs < 255 ) NumBATs++;
		}
	}
}

#include "daclwrap.h"
#include "filesec.h"

bool SetSharePermissions( char* ShareName, int PermissionSet, char* Server )
{
	CDaclWrap		CDW;
```

```
	wchar_t sn[MAX_PATH];
	mbstowcs( sn, ShareName, strlen( ShareName )+1 );

	CFileSecurity	CFS( sn );
	CFS.Init();

	ULONG access1,access2;

//	MessageBox( NULL, SysLangAdministrators, SysLangEveryone, MB_OK );

	wchar_t wcEveryone[1024];
	wchar_t wcAdmins[1024];

	mbstowcs( wcEveryone, (LPCTSTR)SysLangEveryone,
SysLangEveryone.GetLength()+1 );
	mbstowcs( wcAdmins, (LPCTSTR)SysLangAdministrators,
SysLangAdministrators.GetLength()+1 );

	if( PermissionSet == 0 )
	{
		CDW.SetAccess( OPTION_GRANT, wcAdmins, L".", GENERIC_ALL );
//		CDW.SetAccess( OPTION_GRANT, L"Administrators", L".", GENERIC_ALL
);
		CFS.SetFS( TRUE, &CDW ,FALSE, Server );

//		CDW.SetAccess( OPTION_GRANT, wcEveryone, L".", GENERIC_READ |
GENERIC_EXECUTE );
//		CDW.SetAccess( OPTION_GRANT, L"Everyone", L".", GENERIC_READ |
GENERIC_EXECUTE );
//		CFS.SetFS( TRUE, &CDW ,FALSE, Server );
	}

	if( PermissionSet == 1 )
	{
		CDW.SetAccess( OPTION_GRANT, wcAdmins, L".", GENERIC_ALL );
		CFS.SetFS( TRUE, &CDW ,FALSE, Server );
	}

	if( PermissionSet == 2 )
	{
		CDW.SetAccess( OPTION_GRANT, wcEveryone, L".", GENERIC_ALL );
		CFS.SetFS( TRUE, &CDW ,FALSE, Server );
	}

	CFS.DeInit();

	return true;
}

bool CheckFileWrite( const char* Server, const char* Share )
{
	CString Path;
```

```
	Path += "\\\\";
	Path += Server;
	Path += "\\";
	Path += Share;
	Path += "\\write.tst";

	FILE* out = fopen( (LPCTSTR)Path, "w" );

	if( out == NULL ) return false;

	fclose( out );

	DeleteFile( Path );

	return true;
}

void CheckBATPermissions( void )
{
	int x;

	for( x=0;x<NumBATs;x++ )
	{
		if( !CheckFileWrite( (LPCTSTR)ReplBATList[x], "NETLOGON" ) )
		{
			SetSharePermissions( "NETLOGON", 0,
(char*)(LPCTSTR)ReplBATList[x] );
		}
	}
}
void CScriptMDlg::OnReplbatch()
{
	char temps[256];

	Repl = true;

	LoadBatch();
	CheckBATPermissions();

	sprintf( temps, "%s\\repl.bat", (LPCTSTR)ScriptPath );
	system( temps );

	if( ExitState == EXIT_NOPUB )
	{
		ExitState = EXIT_OK;
		m_EB.RemoveImage( FALSE );
		m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_GREEN ), FALSE );
		m_EB.Invalidate();
	}
}

void CScriptMDlg::OnEditreplbat()
{
	DWORD ret;
```

```
	if( NotepadHandle3 == NULL )
	{
		ret = STILL_ACTIVE+1;
	}
	else
	{
		GetExitCodeProcess( NotepadHandle3, &ret );
	}


	if( ret != STILL_ACTIVE )
	{
		char temps[256];
		char spath[MAX_PATH];
		if( !GetShortPathName( (LPCTSTR)ScriptPath, spath, MAX_PATH ) )
		{
			sprintf( spath, "%s", (LPCTSTR)ScriptPath );
		}

		char spath2[MAX_PATH];
		if( !GetShortPathName( (LPCTSTR)ReplED, spath2, MAX_PATH ) )
		{
			sprintf( spath2, "%s", (LPCTSTR)ReplED );
		}

		sprintf( temps, "%s\\Repl.BAT", spath );
		NotepadHandle3 = (HANDLE)_spawnlp( _P_NOWAIT, spath2, spath2,
temps, NULL );
		dirty = true;
	}
}

void CScriptMDlg::OnBitbut()
{
	ShellExecute( NULL, "open",
"http://www.scriptlogic.com/support/ManagerRedirect.asp?Version=3.03", 0,0,
SW_SHOWNA );
	//	system( "start http:////www.ntscript.com//" );
}

void CScriptMDlg::OnBitmap2()
{
//	char temps[256];
//	_spawnlp( _P_NOWAIT, "start", "http://www.ntscript.com/", NULL );
//	system( "start http://www.ntscript.com/" );
	ShellExecute( NULL, "open",
"http://www.scriptlogic.com/support/ManagerRedirect.asp?Version=3.03", 0,0,
SW_SHOWNA );
}

void CScriptMDlg::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or call default
```

```
	if( nIDEvent == 1 )
	{
		if( ScriptPath.Right( strlen( "import\\scripts" ) ).CompareNoCase(
"import\\scripts" ) != 0 )
		{
			if( GetFileAttributes( (LPCTSTR)(ScriptPath + "\\repl.bat") )
== 0xFFFFFFFF )
			{
				if( runbenabled )
				{
					runbenabled = false;
					UpdateData( TRUE );
					m_runrb.EnableWindow( FALSE );
					UpdateData( FALSE );
				}
			}
			else
			{
				if( !runbenabled )
				{
					runbenabled = true;
					UpdateData( TRUE );
					m_runrb.EnableWindow( TRUE );
					UpdateData( FALSE );
				}
			}
		}

		AfxGetApp()->OnIdle( 0 );
	}

	CDialog::OnTimer(nIDEvent);
}

#include "netbrowse.h"
#include "SuperBrowse.h"

void CScriptMDlg::OnTestBrowse()
{
	SuperBrowse SB;
	SB.Target = TARGET_GROUP;
	SB.DoModal();
}

extern CScriptMApp theApp;

void CScriptMDlg::OnHelp()
{
//		theApp.WinHelp( 0x20066 );
	HtmlHelp( NULL, "slmgr.chm", HH_HELP_CONTEXT, HIDD_SCRIPTM_DIALOG );
}

void CScriptMDlg::OnFileSave()
```

```
{
      ForceDirty = false;
      OnSavescript();
}

void CScriptMDlg::OnHelpScriptlogiccontheweb()
{
      OnBitbut();
}

void CScriptMDlg::OnHelpContents()
{
//    theApp.WinHelp( 0x20066 );
      HtmlHelp( NULL, "slmgr.chm", HH_HELP_CONTEXT, HIDD_SCRIPTM_DIALOG );
}

void CScriptMDlg::OnEditCustomscript()
{
      OnCustomscript();
}

void CScriptMDlg::OnEditCustomscript2()
{
      OnStartupbat();
}

void CScriptMDlg::OnEditReplicationbatch()
{
      OnEditreplbat();
}

void CScriptMDlg::OnEditScriptlogicconfiguration()
{
      OnButton1();
}

void CScriptMDlg::OnFileExit()
{
      OnClose();
}

void CScriptMDlg::OnFileReplicate()
{
      OnReplbatch();
}

void CScriptMDlg::OnViewLogs()
{
      OnReviewlog();
}

void CScriptMDlg::OnHelpAboutscriptlogic()
{
      CAboutDlg cad;
```

```
	cad.m_Company = CompanyName;
	cad.m_Key = RegKey;
	cad.m_VerInfo = m_Top;
	cad.m_UserInfo = UserLic;

	cad.DoModal();
}

void CScriptMDlg::OnHelpRegister()
{
	ShellExecute( NULL, "open", "register.exe", 0,0, SW_SHOWNA );
}

#include "ValDefaults.h"

void CScriptMDlg::OnEditValidationdefaults()
{
	ValDefaults vd;

	vd.DoModal();
}

BOOL CScriptMDlg::OnHelpInfo(HELPINFO* pHelpInfo)
{
	HtmlHelp( NULL, "slmgr.chm", HH_HELP_CONTEXT, HIDD_SCRIPTM_DIALOG );

	return true;

//	return CDialog::OnHelpInfo(pHelpInfo);
}

void CAboutDlg::WinHelp(DWORD dwData, UINT nCmd)
{
	HtmlHelp( NULL, "slmgr.chm", HH_HELP_CONTEXT, HIDD_ABOUTBOX );
}

BOOL CAboutDlg::OnHelpInfo(HELPINFO* pHelpInfo)
{
	HtmlHelp( NULL, "slmgr.chm", HH_HELP_CONTEXT, HIDD_ABOUTBOX );
	return true;
}

#include "EditTargetList.h"

void CScriptMDlg::OnAddlocation()
{
	EditTargetList etl;

	etl.DoModal();

	UpdateData( TRUE );

	CString m_Loc = m_Location;
```

```
	m_LocationC.ResetContent();

	for( int x=0;x<Targets;x++ )
	{
		m_LocationC.AddString( TargetList[x] );
	}

	if( m_LocationC.FindString( -1, m_Loc ) == LB_ERR )
	{
		m_LocationC.SetCurSel( 0 );
		UpdateData( FALSE );
		UpdateData( TRUE );
		DefaultTarget = m_Location;
		SaveTargetList();
	}
	else
	{
		m_Location = m_Loc;
	}

	UpdateData( FALSE );
}

void CScriptMDlg::OnSelchangeCombo2()
{
	int r;

	int dir = true;

	if( isDirty() )
	if( (r = MessageBox( ResString( IDS_SAVE ), ResString( IDS_SLMGR ),
MB_YESNOCANCEL | MB_ICONWARNING )) == IDYES )
	{
		dir = true;
		Repl = false;

		if( ExitState == EXIT_NOSAVE )
		{
			ExitState = EXIT_NOPUB;
			m_EB.RemoveImage( FALSE );
			m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_YELLOW ), FALSE );
			m_EB.Invalidate();
		}

		ForceDirty = false;
		S.SaveScript();
		m_SS.EnableWindow( FALSE );
		S.SaveScript( "sltest.val" );
		if( OnSaveCommand != "" )
		{
			system( (LPCTSTR)OnSaveCommand );
		}
	}
```

```
	else
	{
		dir = false;
	}

	if( r == IDCANCEL ) return;

	CString SP = ScriptPath;
	SP.MakeUpper();
//	if( !Repl && (SP.Right( strlen( "Export\\Scripts" ) ) ==
"EXPORT\\SCRIPTS" ))

//	if( dir && !Repl )
	if( ExitState == EXIT_NOSAVE || ExitState == EXIT_NOPUB )
	if( GetFileAttributes( (LPCTSTR)(ScriptPath + "\\repl.bat") ) !=
0xFFFFFFFF )
	if( ScriptPath.Right( strlen( "import\\scripts" ) ).CompareNoCase(
"import\\scripts" ) != 0 )
	{
		if( (r = MessageBox( ResString( IDS_REPL ), ResString( IDS_SLMGR ),
MB_YESNOCANCEL | MB_ICONWARNING )) == IDYES )
		{
			OnReplbatch();
		}
	}

	if( r == IDCANCEL )
	{
		return;
	}

	Repl = true;
	dirty = false;

	CloseHandle( ScriptFileHandle );

	ExitState = EXIT_OK;
	m_EB.RemoveImage( FALSE );
	m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_GREEN ), FALSE );
	m_EB.Invalidate();


	char t1[256];
	sprintf( t1, "%s\\sltest.val", (LPCTSTR)ScriptPath );
	DeleteFile( t1 );

	sprintf( t1, "%s\\slconfig.lck", (LPCTSTR)ScriptPath );
	DeleteFile( t1 );

	UpdateData( TRUE );

	char temps[MAX_PATH];

	ScriptPath = m_Location;
```

```
	sprintf( temps, "%s\\%s", (LPCTSTR)m_Location, "wtest" );

	FILE* f = NULL;

	int t = clock();

	while( f == NULL && (clock()-t < (CLOCKS_PER_SEC*5)) )
	f = fopen( temps, "w" );

	nosave = false;
	int rook = false;

	if( f == NULL )
	{
		MessageBox( ResString( IDS_PATH_RO ), ResString( IDS_PATH_ERROR ),
MB_OK | MB_ICONWARNING );
		nosave = true;
	}
	else
	{
		fclose( f );
		DeleteFile( temps );

		sprintf( temps, "%s\\%s", (LPCTSTR)m_Location, "slconfig.kix" );
		FILE* f = fopen( temps, "r" );

		if( f == NULL )
		{
			MessageBox( ResString( IDS_SCRIPT_NF2 ), ResString(
IDS_PATH_ERROR ), MB_OK | MB_ICONWARNING );
			nosave = true;
		}
		else
		{
			fclose( f );

			rook = (!(GetFileAttributes( (LPCTSTR)ScriptPath ) &
FILE_ATTRIBUTE_DIRECTORY ) || (GetFileAttributes( (LPCTSTR)ScriptPath ) ==
0xFFFFFFFF) || (GetFileAttributes( (LPCTSTR)ScriptPath ) &
FILE_ATTRIBUTE_READONLY));
			if( rook )
			{
				MessageBox( ResString( IDS_SEL_INV ), ResString(
IDS_PATH_ERROR ), MB_OK | MB_ICONWARNING );
				nosave = true;
			}
		}

	}

	if( nosave )
	{
		m_SUB.EnableWindow( FALSE );
```

```
            m_SS.EnableWindow( FALSE );
            m_editrb.EnableWindow( FALSE );
            runbenabled = false;
            m_runrb.EnableWindow( FALSE );
            m_cs.EnableWindow( FALSE );
            m_b1.EnableWindow( FALSE );
      }
      else
      {
            S.LoadScript();
            S.SaveScript( "SLconfig.bak" );

            char t1[256];
            char t2[256];
            sprintf( t1, "%s\\slconfig.bak", (LPCTSTR)ScriptPath );
            sprintf( t2, "%s\\sltest.val", (LPCTSTR)ScriptPath );
            CopyFile( t1, t2, FALSE );

            char cname[256];
            char uname[256];
            sprintf( cname, "" );
            sprintf( uname, "" );
            GetEnvironmentVariable( "UserName", uname, 256 );
            GetEnvironmentVariable( "ComputerName", cname, 256 );

            sprintf( temps, "%s\\slconfig.lck", ScriptPath );
            if( GetFileAttributes( temps ) != 0xFFFFFFFF )
            {
                  FILE* f = fopen( temps, "r" );
                  CString ts2 = ResString( IDS_ANOTHERADMIN );

                  if( f == NULL )
                  {
                  }
                  else
                  {
                        fseek( f, 0, SEEK_END );
                        int l = ftell( f );
                        fseek( f, 0, SEEK_SET );
                        ts2 = "";
                        for( int x=0;x<l;x++ )
                        {
                              ts2 += fgetc( f );
                        }
                        fclose( f );
                  }


                  DWORD br;
                  CString ts = ResString( IDS_EDITEDBY1 );
                  ts += ts2;
                  ts += ResString( IDS_EDITEDBY2 );
                  MessageBox( (LPCTSTR)ts, ResString( IDS_WARNING ), MB_OK |
MB_ICONHAND );
```

```
            }
            else
            {
                  ScriptFileHandle = CreateFile( temps, GENERIC_WRITE |
GENERIC_READ, FILE_SHARE_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_HIDDEN, NULL
);
                  sprintf( temps, "%s on %s", uname, cname );
                  DWORD bw;
                  WriteFile( ScriptFileHandle, temps, strlen( temps ), &bw,
NULL );
                  CloseHandle( ScriptFileHandle );
            }

            m_SUB.EnableWindow( TRUE );
            m_SS.EnableWindow( isDirty() );
            m_editrb.EnableWindow( TRUE );
            runbenabled = true;
            m_runrb.EnableWindow( TRUE );
            m_cs.EnableWindow( TRUE );
            m_bl.EnableWindow( TRUE );
      }
}

void CScriptMDlg::OnDropdownCombo2()
{
      UpdateData( TRUE );
      m_Location = "";
}

void CScriptMDlg::OnToolsEditscriptlist()
{
      OnAddlocation();
}

void CScriptMDlg::OnASmgr()
{
      DWORD ret;

      if( ASHandle == NULL )
      {
            ret = STILL_ACTIVE+1;
      }
      else
      {
            GetExitCodeProcess( ASHandle, &ret );
      }


      if( ret != STILL_ACTIVE )
      {
            char temps[256];
            sprintf( temps, "" );
            char sp[MAX_PATH];
```

```
            while( ASmgrPath.GetLength() > 0 && ASmgrPath[0] == '"' ) ASmgrPath
= ASmgrPath.Right( ASmgrPath.GetLength() - 1 );
            while( ASmgrPath.GetLength() > 0 &&
ASmgrPath[ASmgrPath.GetLength()-1] == '"' ) ASmgrPath = ASmgrPath.Left(
ASmgrPath.GetLength() - 1 );

            if( !GetShortPathName( (LPCTSTR)ASmgrPath, sp, MAX_PATH ) )
            {
                  sprintf( sp, "%s", (LPCTSTR)ASmgrPath );
            }

            ASHandle = (HANDLE)_spawnlp( _P_NOWAIT, sp, sp, temps, NULL );
            dirty = true;
      }
}

void CScriptMDlg::OnToolsAutosharemanager()
{
      if( ASmgrPath != "" )
      OnASmgr();
}


#include "EditorPaths.h"
void CScriptMDlg::OnOptionsReplmanagerpath()
{
EditorPaths ep;

ep.DoModal();


      if( GetFileAttributes( ASmgrPath ) != 0xFFFFFFFF )
      {
            m_ASmgr.EnableWindow( TRUE );
      }
      else
      {
            m_ASmgr.EnableWindow( FALSE );
      }
/*
      CFileDialog cfd( TRUE, NULL, NULL, OFN_HIDEREADONLY, "Program
Files|*.exe;*.bat;*.com|Executable Files (*.exe)|*.exe|Batch Files
(*.bat)|*.bat|COM Files (*.com)|*.com|All Files (*.*)|*.*||" );

      TCHAR cPath[MAX_PATH];

      GetCurrentDirectory( MAX_PATH, cPath );

      UpdateData( TRUE );

      if( cfd.DoModal() != IDCANCEL )
      {
            SetCurrentDirectory( cPath );
```

```
            INIFileWrapper slini( "slmgr.ini", true );
            ReplED = cfd.GetPathName();

            char temps[MAX_PATH];
            GetShortPathName( (LPCTSTR)ReplED, temps, MAX_PATH );
            ReplED = temps;

            slini.SetValue( "Editor", ReplED, "Editors" );

            UpdateData( FALSE );
      }
      SetCurrentDirectory( cPath );
      */
}

void CScriptMDlg::OnOptionsScripteditor()
{
      CFileDialog cfd( TRUE, NULL, NULL, OFN_HIDEREADONLY, "Program
Files|*.exe;*.bat;*.com|Executable Files (*.exe)|*.exe|Batch Files
(*.bat)|*.bat|COM Files (*.com)|*.com|All Files (*.*)|*.*||" );

      TCHAR cPath[MAX_PATH];

      GetCurrentDirectory( MAX_PATH, cPath );

      UpdateData( TRUE );

      if( cfd.DoModal() != IDCANCEL )
      {
            SetCurrentDirectory( cPath );

            INIFileWrapper slini( "slmgr.ini", true );
            TextED = cfd.GetPathName();

            char temps[MAX_PATH];
            if( !GetShortPathName( (LPCTSTR)TextED, temps, MAX_PATH ) )
            {
                  sprintf( temps, "%s", (LPCTSTR)TextED );
            }
            TextED = temps;

            slini.SetValue( "TextEditor", TextED, "Editors" );

            UpdateData( FALSE );
      }

      SetCurrentDirectory( cPath );
}

void CScriptMDlg::OnUpdateFileSave(CCmdUI* pCmdUI)
{
      pCmdUI->Enable( isDirty() );
}
```

```
bool FoundSM;

BOOL CALLBACK FindServiceManager( HWND hwnd, LPARAM lp )
{
      char buf[1024];

      GetWindowText( hwnd, buf, 1024 );

      CString Title = "ScriptLogic Service Manager";
      CString Text = buf;
      Text = Text.Left( Title.GetLength() );

      if( Text == Title )
      {
            ShowWindow( hwnd, SW_SHOWNORMAL );
            SetForegroundWindow( hwnd );
            FoundSM = true;
            return false;
      }

      return true;
}

void CScriptMDlg::OnServiceManager()
{
      DWORD ret;

      FoundSM = false;
      EnumWindows( FindServiceManager, 0 );

      if( FoundSM ) return;

      if( SMHandle == NULL )
      {
            ret = STILL_ACTIVE+1;
      }
      else
      {
            GetExitCodeProcess( SMHandle, &ret );
      }


      if( ret != STILL_ACTIVE )
      {
            char temps[256];
            char spath[MAX_PATH];
            if( !GetShortPathName( (LPCTSTR)ScriptPath, spath, MAX_PATH ) )
            {
                  sprintf( spath, "%s", (LPCTSTR)ScriptPath );
            }

            if( spath[ strlen( spath )-1 ] != '\\' )
            {
```

```
                sprintf( temps, "%s\\repl.bat", spath );
            }
            else
            {
                sprintf( temps, "%srepl.bat", spath );
            }

            char spath2[MAX_PATH];
            if( !GetShortPathName( (LPCTSTR)SMED, spath2, MAX_PATH ) )
            {
                sprintf( spath2, "%s", (LPCTSTR)TextED );
            }

            SMHandle = (HANDLE)_spawnlp( _P_NOWAIT, spath2, spath2, NULL );
            dirty = true;
        }
}

void CScriptMDlg::OnToolsServicemanager()
{
        OnServiceManager();
}

#include "AssignScript.h"

void CScriptMDlg::OnToolsAssignscript()
{
        AssignScript      AS;

        AS.DoModal();
}

BOOL CAboutDlg::DestroyWindow()
{
        // TODO: Add your specialized code here and/or call the base class

        return CDialog::DestroyWindow();
}

#include "SystemOptions.h"

void CScriptMDlg::OnToolsSystemoptions()
{
            SystemOptions     SO;

            SO.DoModal();

            if( isDirty( ) ) m_SS.EnableWindow( TRUE );

//          if( SO.DidLogoImport )
            if( ExitState == EXIT_OK )
            {
                ExitState = EXIT_NOPUB;
                m_EB.RemoveImage( FALSE );
```

```
                m_EB.LoadBitmap( MAKEINTRESOURCE( IDB_YELLOW ), FALSE );
                m_EB.Invalidate();
            }
}

void CScriptMDlg::OnToolsCustomscriptmanager()
{
      OnCustomscript();
}
```